VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZACE IMPORTU MIB SOUBORŮ

AUTOMATION OF MIB FILES IMPORT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. ADAM KOŽUŠNÍK**
AUTHOR

VEDOUCÍ PRÁCE **doc. RNDr. JITKA KRESLÍKOVÁ, CSc.**
SUPERVISOR

BRNO 2018

Abstrakt

Tato diplomová práce se zabývá automatizací procesu importování MIB souborů do MIB databáze ve firmě SolarWinds. Cílem je analyzovat současný postup, kterým je importování prováděno, dále navrhnout změny stávajících nástrojů pro použití při automatizaci, případně navrhnout zcela nové nástroje, pokud to bude nutné. Nakonec navrhnutý systém implementovat v požadovaném programovacím jazyce za použití vhodných technologií. Vše bude prováděno za průběžných konzultací se zodpovědnou osobou firmy SolarWinds.

Abstract

This diploma thesis is about an automation of importing MIB files into MIB database. This work is written for SolarWinds company. The goal is to analyze current process of importing MIB files. Than design changes of existing tools for their use in the automation. Also design new tools, if they are needed. Designed system will be implemented in required programming language with suitable technologies. There will be consultations with responsible person from SolarWinds company during the whole work.

Klíčová slova

Automatizace, SNMP, MIB, C#, .NET, ASP.NET, Angular, HTML

Keywords

Automation, SNMP, MIB, C#, .NET, ASP.NET, Angular, HTML

Citace

KOŽUŠNÍK, Adam. *Automatizace importu MIB souborů*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Jitka Kreslíková, CSc.

Automatizace importu MIB souborů

Prohlášení

Prohlašuji, že jsem tento semestální projekt vypracoval samostatně pod vedením paní doc. RNDr. Jitky Kreslíkové, CSc. Další informace mi poskytli Ing. Adam Schreiber a Ing. Jaroslav Klimíček. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Kožušník
4. května 2018

Poděkování

Rád bych poděkoval mé vedoucí doc. RNDr. Jitce Kreslíkové, CSc., za vedení při vytváření této práce. Dále bych rád poděkoval firmě SolarWinds, která mi umožnila v jejich firemním prostředí tuto práci vytvořit. Poděkování patří také zaměstnancům firmy SolarWinds, Ing. Adamu Schreiberovi a Ing. Jaroslavovi Klimíčkovi, za konzultování a vedení v průběhu celé práce.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá problematikou importů MIB (2.2.5) souborů ze softwarového nástroje JIRA (2.2.6) do databáze s MIB daty ve firmě SolarWinds Czech, s.r.o. Pracovníci této firmy dosud museli MIB soubory importovat do databáze manuálně. Celý postup se skládá z více než dvanácti kroků a je časově náročný. Cílem je analyzovat tento stávající postup a současné nástroje, které zaměstnanci používají. Na základě této analýzy je potřeba navrhnout celý systém a implementovat jej pomocí zadaných technologií. Výsledkem by měl být systém, který umožní celý tento proces maximálně automatizovat. Výstupem této práce je webová aplikace, která umožňuje kompletní nastavení aplikace, zpracovávání všech, nebo vybraných JIRA úkolů. Jednotlivé kroky by měly být samostatně spustitelné. Výsledky by měly být přehledně zobrazovány uživateli přes webové rozhraní.

MIB soubory jsou nedílnou součástí většiny softwarů, které komplexněji sledují provoz na počítačových sítích. Tyto softwary totiž využívají pro analýzu síťového provozu protokol SNMP (2.2), který s MIB soubory pracuje. Ty obsahují informace o objemu vstupních a výstupních dat, vytížení jednotlivých uzlů sítě či komponent uzlů a mnoho dalších informací, které správce sítě může sledovat a analyzovat. Na základě těchto informací se poté může upravovat politika sítě. Čím větší sledovaná síť je, tím více MIB souborů je nutné zpracovávat. Současně s objemem dat roste také časová náročnost jejich zpracování. Pracovníci firmy SolarWinds tak tráví mnoho času právě touto činností. Automatizace je proto zcela logickým a nutným krokem k efektivnější a méně náročné práci. Manuální vstupy uživatele bude nejspíše nutné do jisté míry zachovat, vstupů by však mělo být méně a mělo by se jednat pouze o jednoduché úkony.

Celá práce je rozdělena do několika částí. Kapitola 2 popisuje firmu SolarWinds, protokol SNMP, MIB soubory a také nástroj JIRA, se kterým probíhá komunikace. V následující části 3 je podrobně popsán současný způsob importování MIB souborů, kde jsou popsány jednotlivé kroky, které je nutné při importu provést. Také je zde popis nástrojů, které již má firma SolarWinds k dispozici. Jelikož se výsledný systém bude skládat z několika modulů, které musí různě komunikovat, následuje kapitola věnovaná pouze návrhu výsledného systému 4, ve kterém jsou také stručně popsány technologie, které budou při vývoji použity. Uveden je také návrh webového rozhraní. Následující kapitola 5 je věnována samotné implementaci, která popisuje jednotlivé moduly, nastavení aplikace, a také výsledné webové rozhraní. Mimo samotnou implementaci jsou zde také sekce věnované testování této práce a její spuštění. V závěrečné kapitole 6 jsou zhodnoceny dosažené výsledky a návrhy na možná vylepšení.

Práce navazuje na semestrální projekt, ze kterého byly převzaty kapitoly *Prostředí a protokoly* 2, *Současný stav importu MIB souborů* 3 a kapitola *Návrh* 4. Návrh byl následně

upraven na základě měnících se požadavků a změn vývojového prostředí. Všechny změny jsou uvedeny v kapitole *Implementace* 5.

Kapitola 2

Prostředí a protokoly

V této kapitole je popsáno prostředí, ve kterém by se vytvořený systém používal. Jelikož se jedná o diplomovou práci pro firmu SolarWinds, je zde tato firma krátce zmíněna. Dále je zde popsán protokol SNMP, na kterém je postaveno sledování počítačových sítí a který nasbíraná data uchovává v souborech MIB.

2.1 SolarWinds

Americká firma SolarWinds se sídlem v Austinu ve státě Texas byla založena v roce 1999. Zabývá se především vývojem softwaru určeného pro správu a monitorování počítačových sítí, ve kterém je světovou jedničkou. Dokazuje to mnoho velmi významných zákazníků v čele s NATO, americkou armádou a vládou. Dále také dodávají software pro sledování výkonu různých aplikací, využití výkonu hardwaru či databází a zajištění bezpečnosti počítačových sítí. Valná většina softwaru je placená, některé produkty jsou však dostupné zdarma. Kompletní přehled produktů je k vidění na stránkách firmy SolarWinds[1]. Firma má celosvětově přes 20 poboček. Jednou z nich je i pobočka v Brně s více než třemi sty zaměstnanci. Nejdůležitějším a nejvýnosnějším produktem této firmy, díky němuž firma vznikla, je nástroj Network Performance Monitor, který monitoruje počítačové sítě. Tento produkt, včetně několika dalších, se vyvíjí právě v Brně a podílí se na něm přes dvacet vývojářů. Právě pro tento produkt je tato diplomová práce určena, neboť monitorování sítí je postaveno na protokolu SNMP, který nasbíraná data uchovává v MIB souborech. Tento nástroj využívá přes 20 000 zákazníků, tedy objem dat, který je nutné zpracovat, je opravdu velký. Z tohoto důvodu by automatizace importu MIB souborů byla velkým přínosem a mohla by ušetřit spoustu času.

2.2 SNMP

Základy protokolu SNMP [4], [5] (Simple Network Management Protocol) byly vytvořeny v roce 1988. Jedná se o protokol, který umožňuje řízení a monitorování zařízení v počítačových sítích. Základem protokolu je set operací, který umožňuje administrátorovi sítě měnit statusy SNMP zařízení. Jedná se například o příkazy k vypnutí nějakého rozhraní na směrovačích a přepínačích či zjištění rychlosti, jakou ethernetové rozhraní pracuje. Monitorování však není omezeno pouze na samotný chod sítí. Sledovat lze např. také teplotu

[1] https://www.solarwinds.com/

jednotlivých komponent na zařízeních a varovat tak administrátora v případě jejich přehřívání. Získání informací o využívání procesoru, operační paměti či zaplnění disku je také možné. Protokol Simple Gateway Management Protocol, který byl předchůdcem SNMP, byl navržen pro řízení internetových přepínačů. Naproti tomu SNMP může být použito jak na systémech Windows, tak na systémech Unix, dále na tiskárnách, regálech s modemy, napájeních atd. Nejedná se však pouze o fyzická zařízení. Jakýkoli software, který umožňuje vyhledávání SNMP informací, může být sledován. To samé platí i pro webové servery a databáze. Využití tohoto protokolu je tedy velice široké a nabízí nepřeberné množství možností.

2.2.1 Základní prvky SNMP

Protokol SNMP se skládá z těchto prvků: SNMP agenta, řídící stanice (Network Management Station, zkráceně NMS) a MIB databáze. Ještě je zapotřebí také samotný protokol SNMP pro přenot zpráv od agentů k řídící stanici.

2.2.2 Řídící stanice

Řídící stanice je software, který běží na serveru. Umožňuje komunikaci se SNMP agenty a získává od nich informace o aktuálním stavu na monitorovaných zařízeních. Tomuto procesu se říká „Polling“. Tato data poté mohou být použita pro statitistiky, pro zjišťování problémů na síti nebo ke zjišťování, z kolika procent je agent využíván. Standardní komunikace mezi agentem a řídící stanicí probíhá synchronně. Řídící stanice vyšle dotaz agentovi, který na něj odpoví. V případě nějaké nečekané události, např. výpadku spojení na lince, agenti asynchronně zasílají zprávu „Trap“. Na základě této zprávy a informacích v ní se řídící stanice zachová. Například pokud primární cesta ze sítě do internetu nefunguje, router to oznámí NMS, která to oznámí administrátorovi. Ten následně může potřebná zařízení překonfigurovat tak, aby komunikace opět fungovala.

2.2.3 SNMP Agent

Tento agent je pouze software, který běží na nějakém síťovém zařízení. Buďto se jedná o samostatný program, nebo tento software může být přímo součástí operačního systému, jako je to např. u Cisco IOS na směrovačích. Většina dnešních síťových zařízení v sobě SNMP agenta má. Agent tedy poskytuje sledované informace o zařízení a případné informování o nějaké mimořádné události. Někteří agenti také zasílají zprávy „all clear“, které indikují změnu z chybového stavu na stav, kdy je vše opět v pořádku. Zasílání dat a chybových zpráv se může provést kdykoli, není zde žádné omezení.

2.2.4 Protokol SNMP

SNMP protokol pracuje nad UDP, není zde tedy navázání spojení ani potvrzení přijetí dat. SNMP aplikace tak sama musí zjistit, zda došlo ke ztrátě paketů a případně je odeslat znovu. Řešením je často použití časového limitu. Řídící stanice odešle požadavek na data od agenta. Pokud časový limit vyprší a data nedorazila, dojde ke znovuzaslání žádosti. Kolikrát dojde ke znovuzaslání žádosti o data záleží na nastavení. Po všech pokusech o získání dat je jisté, že došlo k nějakému problému a UDP tak není nijak výrazně limitující. Ovšem v případě chybových zpráv je situace odlišná. Agent, který chybovou zprávu odešle, neví, zda dorazila, a řídící stanice zase neví, že k odeslání došlo. Řešením je použití spolehlivého protokolu

TCP. To se však doporučuje pouze ve speciálních případech. Důvodem, proč je ve většině případů používáno zasílání dat pomocí UDP, je nízké zatížení sítě. TCP totiž nejdříve naváže spojení a poté zasílá data. Každý paket přitom musí být potvrzen, což zvyšuje nároky na propustnost sítě. Navíc UDP umožňuje zasalat zprávu broadcast všem agentům najednou.

2.2.5 MIB

MIB ([3], [2]) neboli Management Information Base je ASCII textový soubor, který popisuje jednotlivé elementy SNMP sítí jako seznam datových objektů. Lze si je představit jako slovník jazyka SNMP, kdy všechny objekty, které jsou uvedeny v SNMP zprávě, musí být dohledatelné v MIB. Hlavním účelem MIB je překlad číselných řetězců do textu čitelného lidmi. Pokaždé, když SNMP agent zašle zprávu, jsou všechna data objektů identifikována pomocí OID (object identifier), které musí každý objekt mít. Řídící stanice SNMP potřebuje znát tyto OID, aby mohla zpracovávat zprávy od SNMP zařízení. Pomocí kompilace pak MIB soubory importuje převedením z ASCII do binárního formátu. Pokud zařízení v SNMP síti není v MIB zapsáno, není možné jej pomocí SNMP sledovat.

MIB soubory jsou psány pomocí notace ASN.1 (Abstract Syntax Notation number One). Tato notace je nezávislá na platformě. Pomocí ní je možné komunikovat mezi rozdílnými systémy. Tato notace je také rozšiřitelná, díky tomu můžeme definovat vlastní objekty. Navíc při definování nového termu můžeme tento term využít jako stavební blok pro další termy, jak můžeme vidět na obrázku 2.1, což je velice užitečné.

```
Letter ::= SEQUENCE     {
      opening     OCTET STRING,
      body        OCTET STRING,
      closing     OCTET STRING,
      address     AddressType
}
```

```
AddressType ::= SEQUENCE      {
      name        OCTET STRING,
      number            INTEGER,
      street            OCTET STRING,
      city        OCTET STRING,
      state       OCTET STRING,
      zipCode     INTEGER
}
```

Obrázek 2.1: Příklad termů v MIB (převzato z [3])

Jak můžeme vidět u termu *Letter*, většina elementů v sekvenci je definovaných pomocí základních elementů *OCTET STRING*. Naproti tomu *AddressType* je definovaný term a samostatná sekvence. Sekvence se tedy může skládat jak ze základních elementů, tak i z dalších sekvencí. Elementy definované v MIB mohou být velice specifické (např. zpráva generována pomocí alarmu na teplotním čidle), nebo naopak velmi obecné.

Jak je již zmíněno výše, každý element musí mít své OID. To může vypadat následovně: `1.3.6.1.4.1.2681.1.2.102`. Toto unikátní číslo identifikuje element v SNMP. Každé OID má také lidsky čitelný popisek. Díky OID je možné hodnoty ve zprávách přiřadit daným zařízením a elementům. OID je prakticky druh adresy. Tuto adresu můžeme rozdělit na dvě části. První identifikuje doménu organizace, která OID vydala, druhá část identifikuje objekt ve sledované doméně. MIB objekty jsou ve stromové struktuře. Každé číslo oddělené tečkou reprezentuje větev stromu. V MIB souborech však tato dlouhá čísla neuvidíme. OID objekt totiž musí být definován pouze jednou a použit pro vytváření dalších objektů, jak je popsáno výše. Na obrázku 2.2 lze vidět, jak mohou být elementy definovány v souboru MIB.

```
dpsInc OBJECT IDENTIFIER ::= {enterprises  2682}
dpsAlarmControl OBJECT IDENTIFIER ::= {dpsInc 1}
dpsRTU OBJECT IDENTIFIER ::= {dpsAlarmControl 2}
dpsRTUsumPClr TRAP-TYPE
      ENTERPRISE  dpsRTU
      VARIABLES { sysDescr,        sysLocation,
dpsRTUDateTime }
      DESCRIPTION  "Generated  when  all  points
clear."
      ::= 102
```

Obrázek 2.2: Příklad definice elementů v MIB (převzato z [3])

U každého elementu z OID 1.3.6.1.4.1.2681.1.2.102 lze spatřit, že každý term vychází z předchozího termu. Na obrázku výše jsou uvedeny poslední čtyři termy uvedeného OID. Term *dpsRTUsumPClr* s číslem 102 (poslední term OID) vychází z *dspRTUN*. Ten má číslo 2 a vychází z termu *dpsAlarmControl*. Takto lze dojít až ke kořenovému termu. Při kompilaci je nutné dodat nejen OID, které si sami definujeme na svých zařízeních, ale také OID veřejných objektů jako jsou: iso, org, internet atd. MIB soubor však nemusí popisovat celý OID strom. Vyšší vrstvy stromu, které definují jeho celkovou strukturu, jsou definovány pomocí standardů RFC. Díky tomu MIB soubor nemusí definovat celý strom. Na začátku každého MIB souboru se nachází část *IMPORTS*, který volá termíny použité v MIB a RFC, které tyto termy definují. Všechny MIB soubory jsou prakticky rozšířením hlavního RFC. Dá se tedy říci, že jednotlivé MIB soubory jsou pouze malé části jednotného Management Information Base. Pokud zařízení podporuje protokol SNMP, mělo by mít svůj vlastní MIB soubor vytvořený výrobcem. Zde pak najdeme všechny elementy, které jsou podporovány pro monitorování a řízení.

Správně napsaný MIB soubor je rozdělen do několika částí oddělených komentáři popisující danou část. Nejdůležitějšími částmi jsou:

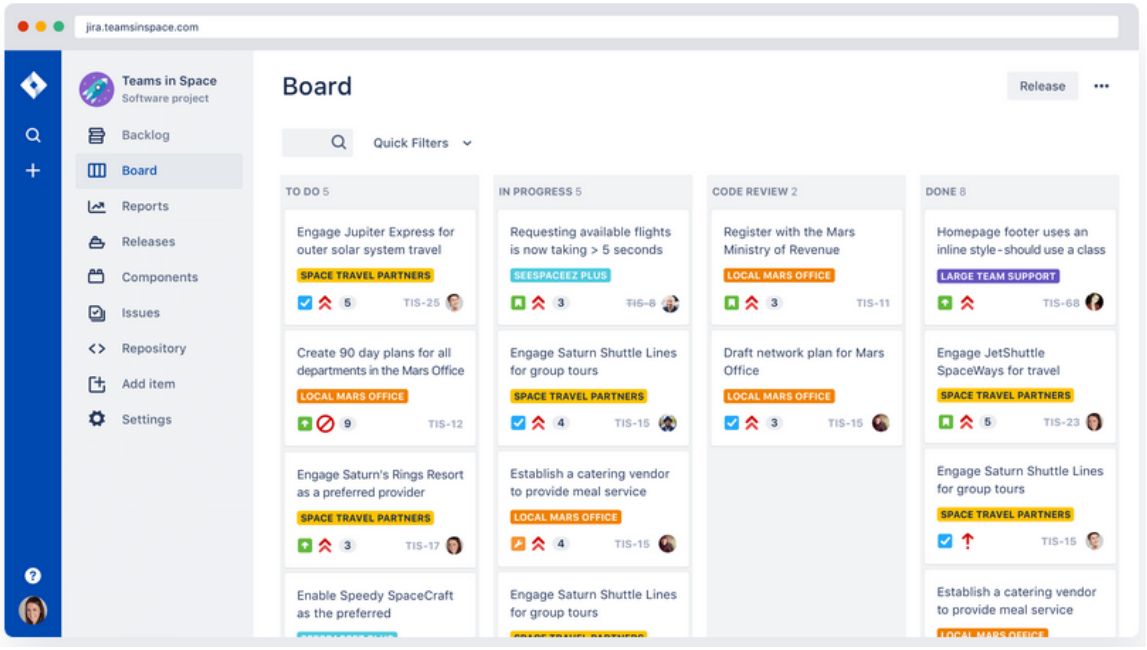
- RFC MIB, které podporují dané zařízení.
- Události (Traps), které je zařízení schopno hlásit řídící stanici.
- Informace, které je možné si od zařízení vyžádat a které je možné zařízení zaslat. Tato funkcionalita je řízená pomocí SNMP příkazů *GetRequest* a *SetRequest*.

Aby bylo možné zasílat žádosti o data ze zařízení nebo data na zařízení odesílat, musí být zařízení v MIB souboru označeno jako *OBJECT=TYPE*. Dále je zde uveden *SYNTAX, ACCESS, STATUS, DESCRIPTION* atd. SYNTAX definuje datový typ, který zařízení poskytuje a přijímá. ACCESS definuje přístupnost zařízení. Jsou zde tři možnosti přístupnosti: nepřístupné, pouze čtení, čtení a zápis. STATUS udává jestli je definice zařízení aktuální, a nakonec DESCRIPTION poskytuje popis dané události.

2.2.6 JIRA

JIRA [1] je softwarový nástroj určený primárně pro řízení vývoje projektů od společnosti Atlassian. Tento nástroj je vhodný zejména pro agilní vývoj softwaru. Poskytuje scrum a kanban nástěnky či zprávy v reálném čase. Na obrázku 2.3 můžeme vidět náhled takové

nástěnky. V levé části obrázku se nachází menu daného projektu. Můžeme zde nalézt backlog, kde se nacházejí úkoly zadané do systému, které si poté jednotlivé scrum týmy mohou vybrat do sprintu. Dále je zde nástěnka, která je na obrázku zobrazena. Obsahuje všechny úkoly, které se týkají daného sprintu (časové období, v němž probíhá vývoj projektu, na jehož konci se zhodnotí, co se během sprintu dařilo, nebo naopak v čem je třeba se zlepšit). Nachází se zde čtyři sloupce:



Obrázek 2.3: Ukázka nástěnky v nástroji JIRA od společnosti Atlassian (převzato z [1])

- TO DO: zde vidíme úkoly, které jsou v momentálním sprintu, ale ještě se na nich nepracuje. Tyto úkoly ještě nemusí být přiřazeny vývojáři.
- IN PROGRESS: zde se nachází úkoly, na kterých se již pracuje. Tyto úkoly již musí být přiřazeny vývojáři, který na nich pracuje.
- CODE REVIEW: tyto úkoly jsou již hotové, musí však ještě projít schválením ostatních členů týmu. Když vývojář úkol dokončí, přidá zde členy svého týmu, aby posoudili jeho práci. Ostatní členové týmu mohou najít chybu, které si vývojář nevšiml. Také mohou znát lepší řešení daného problému. Tento úkon tedy nemá pouze účel kvalitativního posouzení, nýbrž také učící. Všichni členové týmu se totiž mohou zapojit a přiučit se novým technikám a praktikám.
- DONE: úkoly, které jsou hotové a byly schváleny.

Dále v hlavním menu zobrazeném na levé straně můžeme vidět záložky pro výkazy, komponenty, nastavení a další. Jak jsem již zmínil výše, úkoly jsou přiřazovány vývojářům (vybírány vývojáři). Avatar vývojáře, který si úkol vybral, je vidět v pravém spodním rohu každého úkolu. Dále ve spodní části vidíme 3 malé ikony. První zleva určuje typ úkolu. Může se jednat například o úkol opravující chybu, přidávající funkcionalitu nebo vylepšování kódu. Prostřední ikona symbolizuje prioritu úkolu. Na úkolech s vyšší prioritou

by se mělo pracovat co nejdříve. Nejvyšší prioritu by měly mít úkoly označeny jako *Blocker*. Tyto úkoly (na obrázku 2.3 označeny červeným přeškrtnutým kolečkem) blokují další práci, proto by měly být vypracovány jako první. Poslední ikonou je číslo v kolečku. Toto číslo se označuje jako *Story points*. Tímto číslem jsou úkoly ohodnocovány a vyjadřují tak náročnost daného úkolu. Toto je důležité zejména proto, aby tým věděl, jak dlouho mu budou trvat všechny úkoly ve sprintu a zda je mohou stihnout. Takto plánují, jestli úkol za dobu sprintu stihnou dokončit, či je třeba jej např. předat jinému týmu, či nechat na příští sprint. Dále má každý úkol svůj název a status (v barevném rámečku u každého úkolu). Každý úkol může obsahovat spoustu dalších parametrů jako např. verzi softwaru, komponenty, popis, připojené úkoly a další. Všechny parametry daného úkolu se zobrazí po kliknutí na něj. Úkoly je možné také komentovat a vytvořit tak diskuzi při řešení nějakého problému. Pokud je úkol příliš rozsáhlý, je možné jej rozdělit na několik menších podúkolů, což umožňuje lepší spolupráci v týmu a rychlejší vývoj.

Nástroj JIRA má mnoho možností individualizace. Je možné volit mezi různými zobrazeními nástěnek, obrázků pro priority apod. Pracovní postup u úkolů můžeme také přesně definovat. Oproti pracovnímu postupu na obrázku 2.3 můžeme např. odstranit sloupec *Code review* apod. Společnost Atlassian neprodukuje pouze nástroj JIRA, ale také mnoho dalších užitečných nástrojů, které je možné využít při vývoji softwaru. Velmi často firmy využívají také nástroj Confluence nebo Bitbucket. Nástroj Confluence slouží jako místo pro diskuzi a organizaci práce v týmu. Je možné zde zapisovat různé postupy, doporučení či nařízení. Bitbucket slouží jako verzovací systém, kdy uchováváme historii všech provedených změn kódu nebo dat.

Kapitola 3

Současný stav importu MIB souborů

V této kapitole je krok po kroku popsán způsob, jak jsou MIB soubory importovány, co vše je k tomu zapotřebí, a nástroje, které již má firma SolarWinds k dispozici pro importování MIB souborů. Ty jsou poté blíže popsány. Všechny nástroje jsou napsány v jazyce *C# 3* s využitím platformy *.NET 3*, ve kterém bude výsledný program také napsán. Všechny nástroje podrobně zaznamenávají svou práci do souborů. K tomu je použita knihovna *log4net*[1]. Všechny chybové stavy a hlášky je tak možno dohledat. V programech vhodných pro prohlížení takových souborů je možné zprávy vyhledávat nebo filtrovat, a tudíž značně ulehčit zjišťování, co a kde se pokazilo. Nevýhodou však je, že každý nástroj má svůj soubor, kde zapisuje, a proto je nutné kontrolovat každý soubor zvlášť po dokončení daného kroku.

C# a .NET

Většina produktů firmy SolarWinds je napsána v jazyce *C#*, proto také tento nástroj bude implementován v tomto jazyce. Při konzultování práce je možné řešit i ty nejmenší detaily implementace a dosáhnout tak co nejlepšího výsledku, který bude vyhovovat z hlediska korektnosti, spolehlivosti, efektivity a udržovatelnosti. *C#* je vysokoúrovňový objektově orientovaný jazyk, který vyvinula firma Microsoft včetně platformy *.NET*. Jazyk *C#* je založen na jazycích C++[2] a Java[3]. Jedná se o nepřímého potomka jazyka C++, čemuž nasvědčuje velice podobná syntaxe s tímto jazykem. Využití jazyka *C#* je velice rozsáhlé od konzolových aplikací přes databázové programy, až po webové aplikace. Jazyk vznikl v roce 2000, jedná se tak o poměrně nový a moderní jazyk.

3.1 Současný proces importování MIB souborů

Jak jsem již uváděl výše, současný proces importování MIB souborů se provádí manuálně. Zde je seznam kroků, které je nutné provést. Na některé kroky jsou již naimplementovány drobné programy, které ulehčují práci. Zásah pracovníků do celkového průběhu je však značný.

[1] https://www.nuget.org/packages/log4net/
[2] http://www.cplusplus.com/
[3] https://www.java.com/en/

1. Přihlášení se na MIB server, kde se soubory zpracovávají.
2. Zobrazit v softwarovém nástroji JIRA záložku NPM – MIBs for import.
3. Spustit program *JiraDownloader.exe*, který stáhne veškeré MIB soubory do specifikovaného adresáře.
4. Otevřít soubor s chybovými výpisy programu *JiraDownloader.exe* se současným datem. Pokud soubor obsahuje chybové výpisy, které informují o chybějících adresářích, je potřeba uzavřít odpovídající tiket s chybovou hláškou: „Missing directory message template“.
5. Zkopírovat všechny stažené MIB soubory z adresáře pro stahování do pracovního adresáře.
6. Spustit program *ArchiveExtractor.exe* nad pracovním adresářem. MIB soubory od zákazníků přichází v různých archivech. Je proto nutné nejprve všechny archivy extrahovat.
7. Zkontrolovat, zda došlo k extrahování všech archivů. Pokud ne, je nutné je extrahovat manuálně.
8. Spustit nad vyextrahovanými MIB soubory program *MibCompiler.exe*, který je zpracuje. Při kompilování se vyskytují chyby, protože soubory občas bývají poškozeny. Je tedy nutné soubor otevřít ve vhodném editoru, vyhledat příslušný řádek s chybou, který je zapsán v souboru s chybovými výpisy programu *MibCompiler.exe*, a chybu se pokusit opravit.
9. Zálohovat předchozí soubor *MIBsDB.zip* do adresáře s předchozími verzemi.
10. Změnit datum v souboru `c:\mibs\MIBsDB\readme.rtf` na aktuální datum.
11. Spustit *backupfile.bat* pro vytvoření kopie databáze MIB souborů do sdílené složky `\\dev-aus-fs-01\mibshare`.
12. Spustit program *PurgeApp.exe*, který nahradí stávající verzi databáze.
13. Odeslání emailu na oddělení Release Managementu s kopií pro manažera týmu NPM.
14. Zavřít všechny otevřené JIRA tikety na záložce NPM – MIBs for import se zprávou: „Happy scenario message to support“.

3.2 Podrobnější popis jednotlivých kroků

Zde jsou podrobněji popsány kroky, které využívají nějaký nástroj, a jsou už zčásti automatizovány. Kroky jako např. přihlášení na MIB server či kopírování souborů jsou elementární, tudíž zde nejsou popsány. Některé moduly si firma Solarwinds již vytvořila. Je však potřeba je upravit pro automatizaci. Jde zejména o vytvoření rozhraní, přes které bude možné komunikovat a zejména předávat výsledky jednotlivých kroků. Mezi tyto moduly patří *JiraDownloader*, *ArchiveExtractor* a *MibCompiler*. U všech kroků je momentálně nutný manuální zásah uživatele. Každý z následujících nástrojů má svůj konfigurační soubor, ze kterého se načítá nastavení aplikace, a svůj soubor, kam se zapisují všechny provedené události.

3.2.1 JiraDownloader

Tento nástroj je již vytvořený. Jedná se o konzolovou aplikaci, po jejímž spuštění je uživatel vyzván k zadání přihlašovacích údajů. Po jejich potvrzení je uživatel přihlášen do JIRA nástroje a podle nastaveného klíče stáhne všechny JIRA úkoly ze záložky NPM - MIBS for import. Poté stáhne všechny MIB soubory, které jsou k jednotlivým úkolům přidány. Jsou stahovány do složky `c:\mibs\newMibs\<rrmmdd>\<JIRAID_NETSUITID>`, kde <rrmmdd> je aktuální datum ve formátu rok, měsíc, den, které jsou psány za sebou bez mezer a doplněné o JIRA identifikátor a NETSUITE identifikátor. Díky tomuto jednoznačnému pojmenování jsou pak složky s MIB soubory snadno dohledatelné.

Někdy se bohužel stává, že k JIRA úkolu nejsou přiřazeny žádné soubory. Jedná se o chybu na straně zaměstnanců oddělení podpory zákazníků. Je proto potřeba projít soubor `c:\mibs\newMibs\<rrmmdd>\Logs`, kde jsou všechny chybové stavy při stahování MIB souborů zaznamenávány. Pokud se v souboru nachází chyby *Missing directory*, je nutné tyto JIRA úkoly přepnout do stavu Closed s chybovou hláškou: Missing directory message template. Takto se zaměstnancům oddělení podpory zákazníků úkol znovu ukáže, aby mohli chybějící soubory nahrát. Pokud soubor s chybami obsahuje varování, že úkol je již zavřen, je nutné jej zavřít znovu, tentokrát se zprávou *Environmental*.

3.2.2 ArchiveExtractor

Po stáhnutí MIB souborů do pracovního adresáře `c:\mibs\Work\<rrmmdd>` je potřeba soubory extrahovat, neboť jsou různě komprimovány. Tento úkon provádí program ArchiveExtractor, kterým firma SolarWinds již disponuje. Jedná se o různé druhy komprese, např. zip, rar atd. Bohužel tento program nedokáže extrahovat všechny typy těchto souborů a neporadí si taktéž se všemi úrovněmi zanoření souborů do adresářů. Je proto nutné složky projít a zkontrolovat, zda skutečně došlo k rozbalení všech archivů. Jedná se tedy o zbytečně dlouhotrvající krok. Na některých archivech program dokonce zamrzne a celý krok to ještě zkomplikuje.

3.2.3 MibCompiler

Jedná se o jeden ze stěžejních kroků celého procesu importu MIB souborů. Je nutné provést kontrolu MIB souboru, kterou obstarává program MibCompiler. Soubor nesmí obsahovat žádné chyby a musí projít tímto programem bez chyb. Jedná se o konzolovou aplikaci. Případné chyby jsou tak ihned vypsány uživateli. Pokud je soubor zcela v pořádku, dojde ke sloučení současné databáze s MIB daty a právě kompilovaného souboru. Takto jsou nová data vložena do databáze, kterou si zákazník může stáhnout. Program MibCompiler je napsán v jazyce *C#* a pro ověření správnosti dat v souboru využívá knihovnu *ANTLR3*[4].

[4] https://www.nuget.org/packages/Antlr3.Runtime

Kapitola 4

Návrh aplikace

V této kapitole je podrobně popsán návrh celé aplikace. Nejprve je popsán návrh, jak se bude celý proces vykonávat a jaké kroky budou po sobě následovat. Poté budou uvedeny změny, které bude nutné provést na současných nástrojích, které se k importu MIB souborů používají. Následně bude ukázán návrh webového rozhraní, přes které uživatel bude moci celý proces spouštět, upravovat a kontrolovat. Backend celé aplikace bude naprogramován v jazyce *C#*. Webové stránky, které budou rozhraním pro komunikaci s uživatelem, budou napsány v jazyce *HTML*[1] a *Angular4*[2]. Jelikož dojde ke spojení všech současných nástrojů, bude se vytvářet pouze jeden soubor se záznamy celého procesu, což ještě usnadní vyhledávání chyb. Konfigurační soubory se také spojí v jeden soubor. Po konzultaci s pracovníky firmy SolarWinds jsme se dohodli pro využití návrhového vzoru *Dependency injection*[3], přesněji platformu *Castle Windsor*[4].

HTML a Angular4

HTML (HyperText Markup Language) je značkovací jazyk, který slouží k tvorbě webových stránek, které je možné propojit pomocí hypertextových odkazů. Tento jazyk vychází z jazyka SGML (Standard Generalized Markup Language). Dokument napsaný v jazyce HTML se skládá z množiny značek (tzv. tagů) a jejich atributů, které upřesňují jejich vlastnosti. Značky se vkládají mezi úhlové závorky < a >. Každý element dokumentu se skládá z počáteční a ukončovací značky a obsahu. Tím může být např. text v odstavci, jak lze vidět na příkladu: * Ukázka odstavce *. Sémantika dokumentu se určuje právě zanořením jednotlivých značek.

Angular je MVC[5] framework vytvořen a aktualizován firmou Google pro dynamické webové aplikace, kdy se informace v HTML stále mění. Umožňuje vytvářet vlastní HTML elementy a atributy. Obsahuje také *two-way data binding*, kdy dochází k průběžnému obnovování dat mezi *Model* a *View*.

[1] https://html.com/
[2] https://angular.io/
[3] https://msdn.microsoft.com/en-us/library/hh323705(v=vs.100).aspx
[4] http://www.castleproject.org/projects/windsor/
[5] https://www.asp.net/mvc

Dependency injection a Castle Windsor

Dependency injection neboli vkládání závislostí je návrhový vzor používaný v objektově orientovaném programování pro vkládání závislostí mezi jednotlivými komponenty programu. Jednotlivé komponenty tak mohou využívat jiné komponenty, a to i v případě, že v době sestavování programu na ně neměla reference. Lze také říci, že Dependency injection je konkrétnější variantou IoC[6] (Inversion of Control). Závislost je objekt, který může být použit (služba). Vložení je předání tohoto objektu jinému objektu (klientovi), který jej může použít. V praxi to znamená, že objekt využívá služby bez toho, aby byly vytvářeny nové objekty pomocí klíčového slova *new* nebo používání statických metod. Objekt tak přijme vložené hodnoty z vnějšku neboli přenechá vkládání závislostí na někom jiném. Závislosti předává tzv. *dependency provider* neboli kontejner. Objekt tak potřebuje pouze referenci na tento kontejner. Ten je objektu schopen dodat komponenty, které splňují kritéria objektu. Komponenty mohou poskytovat různé služby, v čemž je právě hlavní síla vkládání závislostí. Dalším velkým přínosem je také testovatelnost takto napsaného programu, neboť objektu můžeme předat předem nakonfigurované objekty s námi chtěným chováním.

Pro programování vkládání závislostí lze využít mnoha různých knihoven, např. Castle Windsor, Unity, Ninject či StructureMap. Po konzultaci jsem se rozhodl pro první jmenovanou platformu Castle Windsor, kterou firma SolarWinds hojně využívá ve svých produktech a programech. Jedná se o jednu z nejstarších knihoven pro využití vkládání závislostí napsanou v jazyce C#.

Spuštění

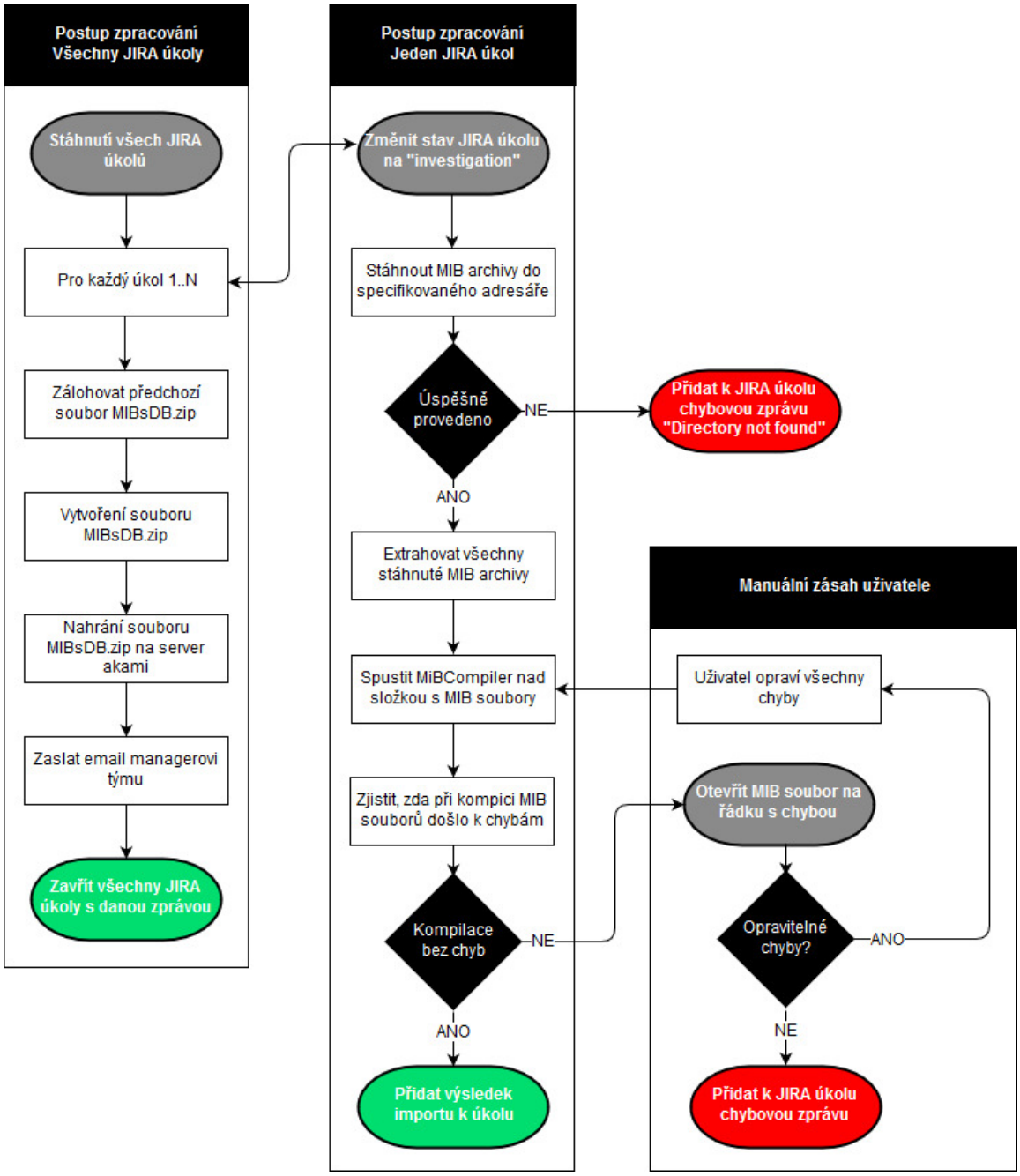
Celý proces má být naprogramován jako služba, u které bude možné naplánovat čas a datum automatického spuštění. Služba si načte všechny potřebné informace z konfiguračních souborů a spustí se. Je to z toho důvodu, aby se celý proces pokusil importovat MIB soubory ještě před příchodem zaměstnanců do práce. Soubory, které budou v pořádku, se naimportují. U ostatních souborů, u kterých se vyskytne chyba, již bude zaznamenána chyba. Zaměstnanci se stačí pouze podívat o jakou chybu se jedná a podle toho provede patřičné kroky, např. opraví chyby v MIB souboru. Díky automatickému spuštění dojde k dalšímu usnadnění práce a snížení času potřebného k naimportování MIB souborů zaměstnancem. Aplikaci bude možno spustit i ručně kdykoli si budeme přát. Taktéž jednotlivé moduly musí být použitelné samostatně.

4.1 Návaznost jednotlivých kroků

Bohužel ani při splnění zadání a dodržení zmíněného návrhu není možné zcela vyloučit manuální zásah uživatele. Ten bude stále potřeba u opravování chyb v MIB souborech, na které upozorní program MibCompiler. Na obrázku 4.1 lze vidět návrh celého procesu zpracování MIB souborů. Návrh je rozdělen do dvou sloupců. Levý sloupec se bude vykonávat pro všechny JIRA úkoly společně, jedná se o stažení úkolů, provedení zálohy databáze apod. Pravý sloupec již popisuje seznam činností, který se bude pro každý JIRA úkol vykonávat samostatně. Zde patří např. stažení MIB archivů, jejich rozbalení a kompilace.

Po přihlášení dojde ke stažení všech JIRA úkolů se všemi potřebnými daty včetně cest k MIB souborům. Přihlášení je nutné ponechat, a to hned z několika důvodů. Bez přihlá-

[6] https://msdn.microsoft.com/en-us/library/ff921087.aspx



Obrázek 4.1: Postup importování MIB souborů (zdroj vlastní)

šení do JIRA nástroje není možné vidět žádné úkoly a provádět jakékoliv změny. Ten, který není zaměstnancem firmy SolarWinds, by neměl mít přístup k těmto citlivým datům jejich zákazníků. Nabízí se možnost použití jednoho univerzálního účtu pro všechny zaměstnance, který by sloužil pouze pro vykonávání tohoto procesu. Tento postup však není možné použít. Je nutné zaznamenávat, kdo dané operace s úkoly prováděl. V případě jakýchkoli nejasností a dotazů se ihned ví, koho je třeba kontaktovat. Navíc firma SolarWinds neumožňuje vytvářet žádné účty navíc. Každý zaměstnanec má z důvodu bezpečnosti pouze jeden účet, který může využívat. Všechny soubory se budou nacházet ve složce, jejíž cesta a jméno se vyčtou z konfiguračního souboru.

Každý JIRA úkol bude dále prováděn zvlášť. Dojde ke změně stavu úkolu na *Investigation*. Poté se stáhnou všechny MIB soubory z cesty uvedené v úkolu. Pokud se stažení nepodaří, je k úkolu přidána zpráva: *Directory not found*. V opačném případě se pokračuje dále. Následuje spuštění nástroje ArchiveExtractor.exe, který rozbalí všechny stažené archivy v adresáři daného úkolu. Nad každým MIB souborem v adresáři se spustí nástroj *MibCompiler*. Ten zjistí, zda v souboru nejsou chyby. Pokud soubor chyby neobsahuje a program se korektně ukončí, dojde ke sloučení tohoto souboru se stávající MIB databází. Pokud však soubor obsahuje chyby, bude otevřen ve vhodném textovém editoru na řádku, kde se chyba nachází. Nabízí se využití programu *Notepad++*. Pokud je chyba opravitelná (např. chyba v gramatice souboru) uživatel ji opraví a spustí program *MibCompiler* na opraveném souboru znovu. Takto bude pokračovat, dokud se v souboru budou vyskytovat chyby, které je možné opravit. Pokud chyba nebude opravitelná, bude k JIRA úkolu přidána příslušná chybová zpráva. Pokud kompilace MIB souboru proběhne v pořádku, je k úkolu přidána zpráva o úspěšném importování MIB souboru do databáze.

Po dokončení výše popsaného postupu na všech JIRA úkolech dojde k zálohování předchozí verze komprimované databáze `c:\mibs\MIBsDB\MIBsDB.zip` do adresáře obsahující tyto archivy. Následně se provede přepsání data v souboru `c:\mibs\MIBsDB\readme.rtf` na aktuální datum a dojde k vytvoření nového archivu celé MIB databáze. Do archivu budou přidány také soubory *MIBs.cfg* a *readme.rtf*. Spuštěním programu *backupfile.bat* se nejnovější databáze nakopíruje do sdíleného adresáře celé firmy SolarWinds. Tento archiv bude následně nahrán pomocí programu `c:\mibs\MIBTools\PurgeApp\PurgeApp.exe` na server firmy SolarWinds, kde si novou verzi databáze mohou zákazníci stáhnout. Nakonec bude zaslán informační email o procesu importu MIB souborů manažerovi týmu a managementu dohlížejícímu na vydávání nových verzí databází a softwaru.

Téměř všechny výše uvedené kroky budou zautomatizovány. Výjimku tvoří krok opravy chyb v MIB souborech, které uživatel bude muset provádět manuálně. Dále bude potřeba zadat správné údaje do konfiguračního souboru. Celý proces bude fungovat jako služba. Nastaví se datum, čas spuštění a bude proveden celý proces. Pokud by proces proběhl v pořádku, došlo by ke značné časové úspoře zaměstnanců firmy SolarWinds. Bohužel takovýchto případů nebude mnoho, neboť velké množství MIB souborů obsahuje chyby, které je nutné opravit. Soubory, které však chyby obsahovat nebudou, budou automaticky přidány do nové MIB databáze. Chybové soubory bude muset uživatel opravit a spustit nad nimi program *MibCompiler.exe*. Dokud neodstraní všechny chyby, které odstranit lze, bude uživatel tento proces opakovat. Bude však už provádět pouze činnosti, které není možné momentálně automatizovat.

4.2 Návrh jednotlivých modulů

Jak již bylo zmíněno v kapitole 3, některé moduly má firma SolarWinds již vytvořené a při importování MIB souborů je zaměstnanci využívají. Je však potřeba tyto moduly upravit tak, aby je bylo možné při automatizaci použít. Každý modul také musí být použitelný zvlášť, a to jako konzolová aplikace. Při automatizaci budou jednotlivé moduly volány z webových stránek. Bude proto nutné použít také vhodné API. Níže budou popsány návrhy změn, které bude nutné provést, a návrhy nových modulů, které se budou muset vytvořit.

4.2.1 ArchiveExtractor

Tento existující modul má mnoho slabin, jak je již uvedeno v kapitole 3.2.2. Při implementaci byla využita knihovna *SevenZipSharp*. Tato knihovna využívá knihovnu *7z.dll*, která je volně dostupná. Tuto knihovnu využívá program *7zip*, který je schopen extrahovat libovolný typ archivu. Bohužel knihovna *SevenZipSharp* je prakticky nástavbou knihovny *7zip* pro použití při programování v jazyce *C#*. Bohužel v *SevenZipSharp* nejsou zastřešeny všechny typy archivů dostupných v knihovně *7z.dll*, a proto není možné rozbalit veškeré archivy. Alternativou by bylo použití jiné knihovny např. *SevenZip*. Nebyla však nalezena žádná volně dostupná knihovna, která by umožňovala extrahování všech archivů, které podporuje knihovna *7z.dll*. Po konzultaci se zaměstnanci firmy SolarWinds jsme došli k závěru, že nejlepší volbou bude využít samotný program *7zip*. Tento program je totiž možné spouštět také přímo v konzoli operačního systému Windows. Je potřeba pouze zadat cestu k archivu, a také adresář, kam se má archiv extrahovat. Je však nutné, aby byl program *7zip* na počítači nainstalován, což bude před samotným spuštěním nutné kontrolovat. Odchytávání výjimek bude také nutné pro případy, kdy daný archiv bude poškozený, nebo nepůjde extrahovat z jakékoliv jiné příčiny. Výhodou tohoto řešení je také způsob, kterým program *7zip* rozpoznává typ archivu. Nejprve se zjistí koncovka archivu a podle ní se pokusí archiv extrahovat příslušným algoritmem. Pokud však dojde k chybě a archiv se nepodaří rozbalit, program *7zip* přečte několik prvních bytů archivu. Každý typ archivu (ale také všech známých typů souborů) je rozeznatelný právě podle prvních několika bytů (liší se u různých souborů, od 1 bytu až po několik desítek bytů). Pokud program *7zip* takto rozpozná typ archivu, extrahuje jej s hláškou, že soubor se nepodařil rozbalit algoritmem odpovídajícímu koncovce archivu, ale podařil se rozbalit algoritmem jiným.

ArchiveExtractor si bude muset poradit s jakoukoli úrovní zanoření adresářů a archivů. Tuto vlastnost používané řešení nenabízelo. V některých případech dokonce došlo k zamrznutí celého programu. Po pečlivém zvážení všech problémů a množství nutných změn bylo rozhodnuto, že lepší variantou se jeví napsat zcela nový program *ArchiveExtractor*, který bude splňovat požadované vlastnosti. Aby jej bylo možné využívat jako konzolovou aplikaci, a také jako modul do automatizace importů MIB souborů, bude nutné oddělit rozhraní modulu od vlastní logiky programu. Vytvořením nového programu se také usnadní implementace s využitím návrhového vzoru Dependency injection. Program totiž bude možné ihned správně navrhnout tak, aby použití *Castle Windsor* bylo co nejjednodušší a bude možné napsat *Unit testy*.

4.2.2 JiraDownloader

Modul *JiraDownloader* je již vytvořený a funguje velice spolehlivě. Bude tedy využit i při automatizaci, musí však projít několika změnami. Tento modul by měl zastřešovat veškerou komunikaci s nástrojem JIRA a nikoli pouze stahování JIRA úkolů. Modul proto bude pře-

jmenován na *JiraCommunicator*. Tento název by měl lépe vystihovat účel celého modulu. Modul bude provádět operace: přihlášení do nástroje JIRA po zadání přihlašovacího jména a hesla, stažení všech JIRA úkolů podle zadaného klíče v konfiguračním souboru, měnění stavu úkolů (zavření úkolu apod.) a stažení archivů s MIB soubory.

Co se týče kódu, je potřeba přidat rozhraní, přes které bude možné s modulem komunikovat. Každý JIRA úkol bude vytvořen podle rozhraní, které určí, jaké atributy úkol bude obsahovat. Jedná se o rozhraní IIsue. Zde je seznam požadovaných atributů:

a) IssueStatus: uchování hodnoty, zda je úkol otevřen, investigován, či zavřen.

b) ExtractorResult: výstup modulu *ArchiveExtractor* obsahující případné chyby nebo informování o správném dokončení modulu.

c) Downloaded: uchovává informaci, zda došlo ke stažení MIB souborů, či nikoli.

d) DownloadError: pokud nastane chyba při stahování MIB souborů, zde je uložena chybová hláška.

e) IOException: zde se uloží zachycená výjimka v případě jakéhokoli pádu.

f) MibsCompiled: ukládá informaci, zda stažené MIB soubory byly zkompilovány v pořádku.

Tyto atributy zachytí všechny důležité události, které se na JIRA úkolu provedou při postupném průchodu jednotlivými kroky automatizace. V tomto modulu dojde k vytvoření listu instancí třídy *JiraIssues*, které ponesou výše uvedené atributy. Tak lze v jednotlivých krocích kontrolovat, zda předchozí krok proběhl v pořádku a je tedy možné provést další krok. Pokud předchozí krok skončí s chybou, tak se tato chyba uchová v příslušném atributu (a také zapíše do souboru s výpisy).

4.3 MibCompiler

MibCompiler je stěžejním modulem celého procesu. Tento modul je již vytvořený. Úpravy, které se zde budou provádět, budou pouze pro poskytnutí rozhraní na použití v automatizaci. Funkčnost zůstane zcela zachována, neboť se jedná o velmi komplexní a dobře fungující program. Stejně jako všechny vytvořené moduly, *MibCompiler* je konzolová aplikace, která zkompiluje zadaný MIB soubor a naimportuje jej do MIB databáze. Pokud soubor nejde zkompilovat, vypíše se na výstup číslo řádku, na kterém se chyba nachází. Tento postup bude zachován také u automatizace. Navíc však bude soubor otevřen ve vhodném textovém editoru s kurzorem na řádku s nalezenou chybou. Nejspíše se bude jednat o editor *Notepad++*[7]. Poté, co uživatel chybu opraví, zadá do konzole opětovné spuštění programu *MibCompiler*. Takto bude pokračovat, dokud soubor nebude obsahovat chyby a půjde zkompilovat, nebo dokud uživatel nenajde chybu, která by byla neopravitelná. V poslední fázi, kdy uživatel bude s programem komunikovat přes webové stránky, bude možné spustit znovu jakoukoli část procesu kliknutím na příslušné tlačítko, což je uživatelsky přívětivější varianta.

[7] https://notepad-plus-plus.org/

4.4 EmailSender

Jak již bylo zmíněno v kapitole 3.1, na konci celého procesu se pošle informativní email manažerovi týmu, který má celý úkol na starost. Zatím zaměstnanci zasílali email ručně. Nyní však dojde k vygenerování a odeslání emailu automaticky. Příjemce emailu a obsah zprávy bude samozřejmě možné změnit v konfiguračním souboru celé aplikace. Nejspíše se nebude jednat o samostatný modul, ale pouze o samostatnou třídu.

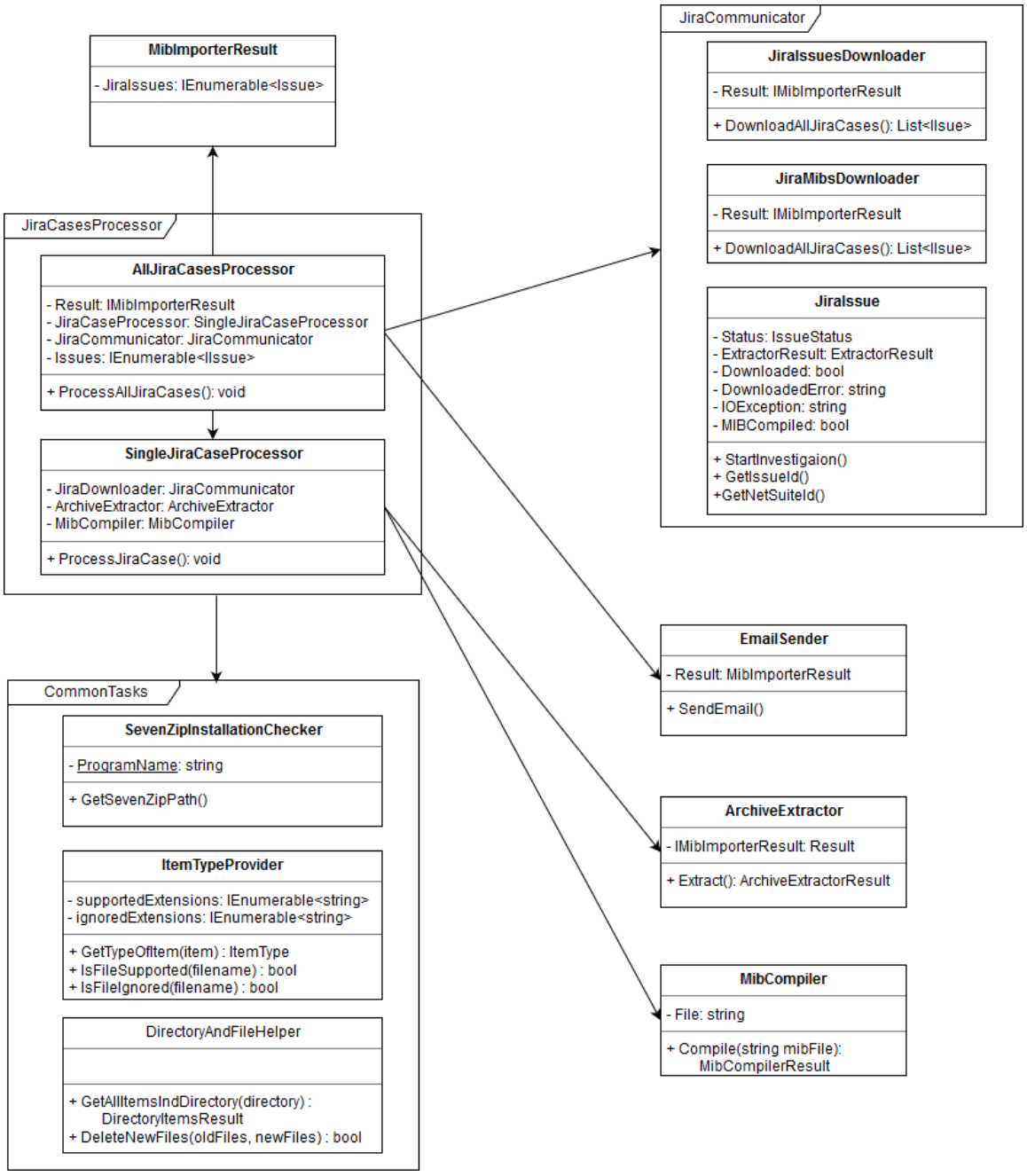
4.5 CommonTasks

V průběhu všech kroků bude potřeba volat různé systémové či vlastní podpůrné akce. Některé se budou používat ve více modulech. Aby nedošlo ke zbytečné duplicitě kódu, rozhodl jsem se vytvořit modul, který bude tyto akce nabízet. Jeho název bude *CommonTasks* a budou zde metody pro procházení složek, porovnání počtu a názvů souborů po nějaké akci (např. po spuštění programu *ArchivExtractor*) atd. Prakticky kterékoliv kroky či metody, jejichž použití se bude vyskytovat na více místech, budou přesunuty do tohoto modulu.

4.6 Návrh pomocí diagramu

Na obrázku 4.2 lze vidět návrh pomocí diagramu (jedná se o kombinaci diagramu tříd a diagramu balíčků). V diagramu jsou zobrazeny pouze nejdůležitější atributy, metody a třídy. Kompletní diagram by totiž byl velice rozsáhlý. Navíc pouze s uvedenými třídami bude možné zvenčí komunikovat. Za těmito třídami bude velké množství dalších tříd, se kterými budou zmíněné třídy spolupracovat a komunikovat.

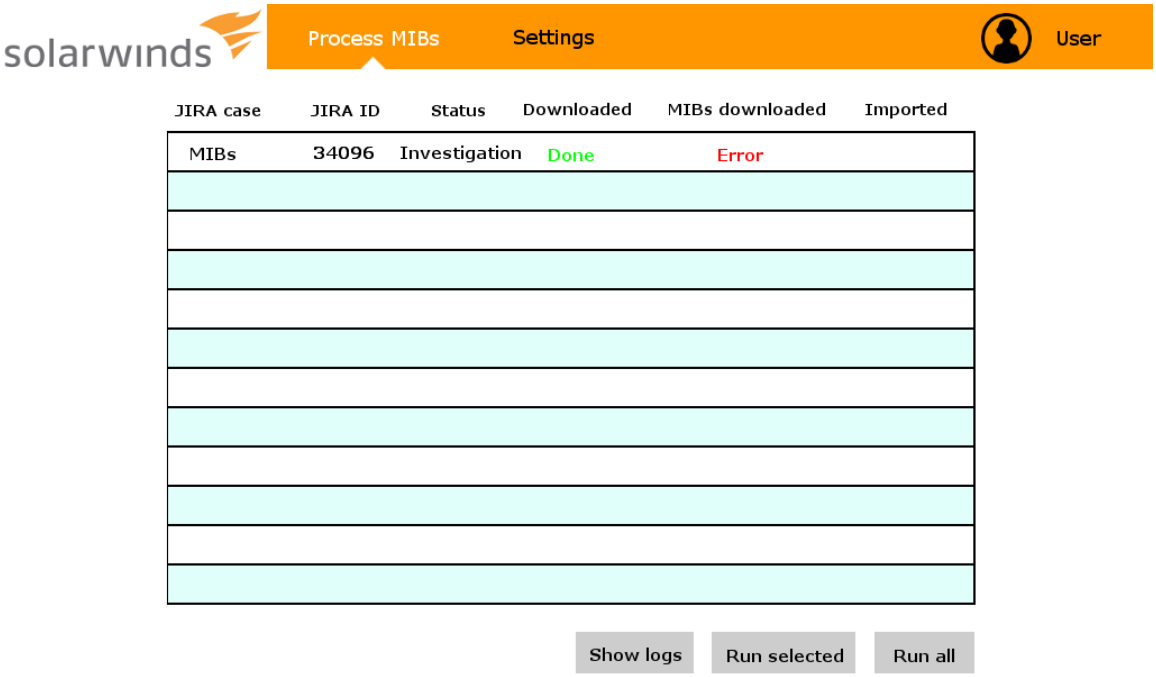
Lze zde jasně vidět, že nejdůležitějším modulem bude *JiraCasesProcessor*, který bude řídit celý průběh vykonávání. Bude zde hlavní metoda, která spustí celý proces. U všech tříd lze také vidět závislosti naznačené pomocí šipek. *JiraCasesProcessor* bude mít pochopitelně závislosti prakticky na všechny uvedené balíčky a třídy. Po stažení všech JIRA úkolů bude postupně na každý z nich zavolána metoda *ProcessJiraCase()* ve třídě *SingleJiraCasProcessor*. Zde se poté budou provádět všechny jednotlivé kroky popsány v kapitole 3.1.



Obrázek 4.2: Diagram tříd a balíčků (zdroj vlastní)

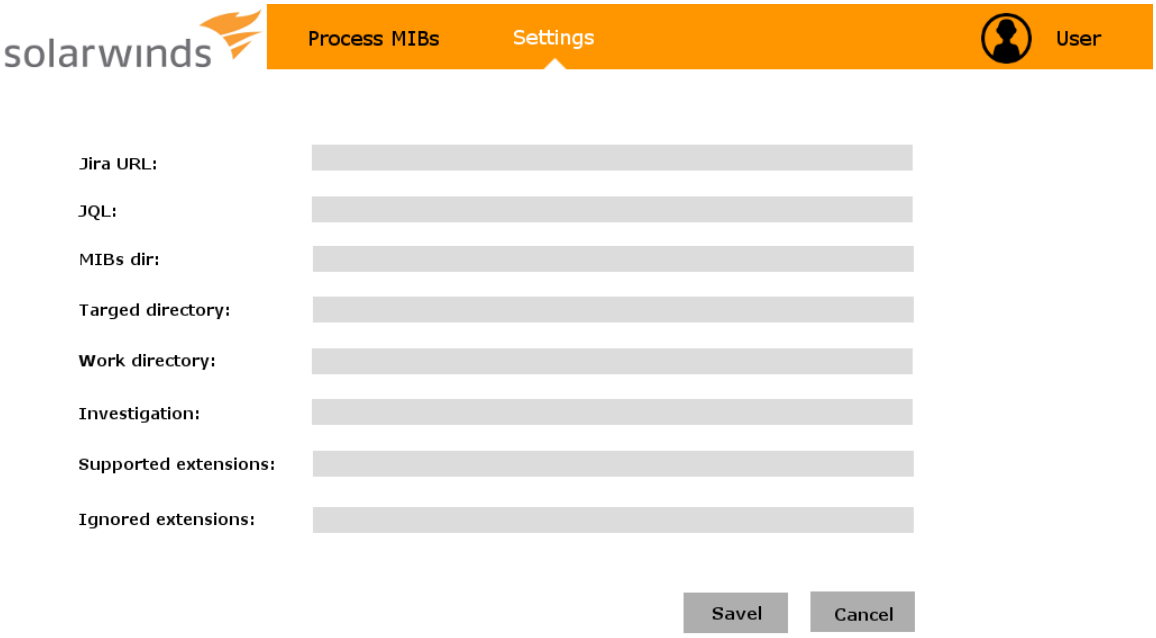
4.7 Webové stránky

Jak již bylo zmíněno výše, tato aplikace poběží na webových stránkách, kde bude moci uživatel vidět JIRA úkoly a jejich stav, ale také spouštět jednotlivé kroky a nastavovat celou aplikaci. Nejprve bude nutné se přihlásit, tudíž se nejprve zobrazí stránka pro zadání přihlašovacího jména a hesla. Po úspěšném přihlášení se zobrazí stránka pro práci. V pravém horním rohu bude zobrazeno jméno přihlášené osoby. Na této stránce budou zobrazeny všechny JIRA úkoly, které se týkají celého procesu. JIRA úkoly budou zobrazeny v tabulce a seřazeny podle toho, jak se stáhnou, čili podle čísla úkolu. Jak lze vidět na obrázku 4.3, u každého úkolu bude zobrazen aktuální stav úkolu a slovní popis *Done* nebo *Error* značící úspěšné provedení daného kroku, resp. chybné provedení daného kroku. Pokud v některém z kroků nastane chyba, další krok se již nevykonává, proto u něj nebude žádný informativní popis. Tento scénář je naznačen na obrázku 4.3, kdy u stažení MIB souborů vidíme *Error*. U následujícího kroku importování tedy žádný popis není. Dále zde budou tři tlačítka. *Run all* spustí celý proces nad všemi JIRA úkoly. Kliknutím na příslušný řádek v tabulce bude možné vybrat daný JIRA úkol a následným kliknutím na tlačítko *Run selected* jej samostatně spustit. Takto bude možné spustit znovu pouze úkoly s chybou. Poslední tlačítko *Show logs* otevře soubor, do kterého se zaznamenávají výpisy jednotlivých kroků aplikace, ať už jde o informační, či chybové výpisy. V tomto souboru bude možné nalézt přesný popis chyb, které nastaly a podle toho zvolit další postup.



Obrázek 4.3: Návrh webové stránky s importováním MIB souborů (zdroj vlastní)

V horní části stránky bude odkaz na webovou stránku, kde se bude měnit nastavení aplikace. Jak bude vypadat stránka s nastavováním aplikace lze vidět na obrázku 4.4. Dostaneme se na ni kliknutím na záložku *Settings*. Jak jsem již zmiňoval výše, bude zde možné nastavit všechny důležité parametry:



Obrázek 4.4: Návrh webové stránky s nastavením aplikace (zdroj vlastní)

1) URL adresa na JIRA server.
2) JQL řetězec, který definuje, jaké JIRA úkoly budou staženy.
3) Cílová složka, kde budou staženy archivy s MIB soubory.
4) Pracovní složka, kde budou kopírovány stažené MIB soubory.
5) Nastavení příznaku o zkoumání JIRA úkolu.
6) Přípony archivů, které jsou podporované a budou se rozbalovat.
7) Přípony souborů, které mají být ignorovány.

Po načtení této stránky se do šedých políček u jednotlivých parametrů načtou jejich hodnoty. Přímo zde je bude možné editovat. Po kliknutí na tlačítko *Save* dojde k uložení nových hodnot parametrů. Pokud klikneme na , dojde ke znovunačtení stávajících hodnot do políček.

Komunikace s webovými stránkami bude implementována pomocí rozhraní REST (Representational State Transfer). Jelikož všechny nástroje musí být spustitelné také samostatně, je nutné u všech vytvořit rozhraní, přes které budou komunikovat. Rozhraní pro komunikaci s webovými stránkami bude muset být také vytvořeno. Rozhraní REST umožňuje jednotný přístup ke zdrojům na serveru, kterými mohou být např. data. REST byl vyvinut společně s protokolem HTTP a právě pomocí něj umožňuje data přenášet. Přenášení dat bude probíhat zejména pomocí metod HttpGet a HttpPost. [6]

Kapitola 5

Implementace

V této kapitole je popsána kompletní implementace celé aplikace. V jednotlivých podkapitolách jsou vždy popsány samostatné moduly. Jednotlivé moduly provádí komunikaci s uživatelem a zápisy do logovacích souborů, extrahování archivů, komunikaci s nástrojem JIRA. Dále je zde modul s algoritmy, které využívají ostatní moduly a také modul, který kompletně řídí celý proces importu MIB souborů, a navíc ještě modul s podpůrnými nástroji pro dokončení importu. Nakonec je zde modul s webových rozhraním, který využívá všechny zmíněné moduly. Aplikace byla implementována v jazyce C#, a to včetně webového rozhraní za použití frameworku .NET verze 4.6.1. U webového rozhraní byly navíc použity frameworky Angular 4 a Bootstrap. Komunikace mezi webovým rozhraním a samotnou logikou aplikace je řešena pomocí REST.

Vytváření instancí

Téměř všechny třídy musí mít rozhraní, a to z důvodu použití *Dependency injection* popsaného výše. Díky tomu při implementaci nebylo nutné vytvářet instance tříd, neboť se o vše postarala knihovna CastleWindsor, která dodává potřebné instance na všechna místa, kde jsou potřebné. Zaregistrování všech potřebných závislostí ve webovém rozhraní je napsáno ve třídě *ServiceResolver*. Pro konzolovou aplikaci jsou závislosti registrovány ve třídě *WindsorInstaller* v modulu JiraCasesProcessor.

5.1 Nastavení aplikace

Nastavení aplikace se načítá ze souboru *app.config*. Tento soubor je v modulu s webovým rozhraním a v modulu, který spouští konzolovou aplikaci. Při kompilaci projektu ve Visual Studiu si každý modul vytvoří svůj vlastní konfigurační soubor, ze kterého by načítal svou konfiguraci. Parametry aplikace však musí být možné nastavovat z webové aplikace. Všechny moduly tedy musí načítat konfiguraci ze stejného konfiguračního souboru. To stejné platí rovněž pro nastavení konfigurace. K tomu bylo využito metody *OpenExeConfiguration()* z knihovny *System.Configuration*. Tato metoda pracuje vždy s konfiguračním souborem ze startovacího modulu, který při překladu vytvoří spustitelný soubor s příponou .exe. Jednotlivým třídám jsou předávána pouze nastavení, která potřebují. Nastavení do tříd je také vkládáno do tříd pomocí *Dependency injection*. Níže jsou uvedeny všechny třídy, které obsahují parametry vyčtené z konfiguračního souboru.

JiraDownloaderSettings

Tato třída uchovává informace potřebné pro komunikaci s nástrojem JIRA v modulu *JiraCommunicator*. Jsou zde pouze tyto dvě hodnoty:

- Jira URL: adresa JIRA serveru firmy SolarWinds.
- Jql: tento řetězec slouží jako filtr pro vybrání všech JIRA úkolů, které souvisejí s importováním MIB souborů.

V tomto projektu je použit následující řetězec Jql pro filtrování JIRA úkolů:

```
project = CUST AND comment = "Area:MIB Submission"AND (Product = "Network
Performance Monitor"OR Product = "Network Configuration Manager"or Product
  = "NetFlow Traffic Analyzer") AND Type = "Customer Feature Request"AND
                  (status != Closed OR updated > -8d)
```

Z výše uvedeného řetězce lze vyčíst, že se musí jednat o úkol od zákazníka (označeno *CUST*) a musí obsahovat komentář *Area:MIB Submission*. Úkol musí být spojený s produktem *Network Performance Monitor*, *Network Configuration Manager*, nebo *Netflow Traffic Analyzer*. Musí být typu *Customer Feature Request*, nesmí se nacházet ve stavu *Closed* a jeho poslední úprava nesmí být starší než 8 dní.

SingleJiraCaseProcessorSettings

V modulu *JiraCasesProcessor* dochází k řízení celého procesu importu MIB souborů. Ve třídě *SingleJiraCaseProcessor* se provádí zpracování jednotlivých JIRA úkolů od stažení souborů z nástroje JIRA, jejich extrakce, až po jejich kompilaci. Jsou zde tedy potřebné zejména cesty k adresářům, ve kterých se MIB soubory v jednotlivých etapách importu nacházejí.

- MibsDir: adresář ve sdíleném úložišti firmy SolarWinds, kde oddělení pro podporu zákazníků ukládá MIB soubory z jednotlivých JIRA úkolů. Odtud jsou MIB soubory stahovány.
- TargetDir: adresář, kde jsou MIB soubory staženy ze sdíleného úložiště na lokální počítač, kde se import provádí.
- WorkDir: pracovní adresář, do kterého jsou MIB soubory nakopírovány z adresáře TargetDir.

CompilerSettings

Tato třída obsahuje nastavení pro MIB kompilátor, který kompiluje MIB soubory a poté je přidává do databáze.

- MibCompilerArgs: argumenty, se kterými má být kompilátor spuštěn. Jsou zapisovány v jednom řetězci a odděleny od sebe mezerou.
- StrAcessCon: řetězec pro připojení k MIB databázi.

MailSenderSettings

Třída s nastavením pro odesílání emailu manažerovi týmu, který s MIB soubory pracuje.

- emailHost: adresa SMTP serveru firmy SolarWinds, ze kterého se odešle email.
- emailAdressReceiver: adresa příjemce emailu.
- emailSubject: předmět emailu.
- emailBodySuccess: tělo emailu v případě, kdy import MIB souborů bude úspěšný.
- emailBodyFail: tělo emailu v případě, kdy import MIB souborů selže.

BackUpDbSettings

Třída s nastavením pro zálohování databáze s nově importovanými MIB soubory.

- BackUpPath: cesta k aplikaci na zálohování MIB databáze.
- BackUpCommand: příkaz ke spuštění zálohování MIB databáze.

PurgeAppCallerSettings

Třída s konfigurací pro nahrání nové MIB databáze na server, odkud si ji mohou stáhnout zákazníci.

- PurgeAppPath: cesta k aplikaci, která nahrává MIB databázi na server pro zákazníky.
- PurgeAppFile: název souboru, který se má spustit pro nahrání MIB databáze na server.

ArchiveCreatorSettings

Třída s nastavením pro vytvoření archivu MIB databáze.

- ArchiveCreationPath: cesta, kde se má archiv MIB databáze vytvořit.
- ArchiveCreationTimeout: časový interval, do kterého se musí archiv vytvořit, jinak je vytváření zastaveno.
- FilesToArchive: názvy souborů oddělené středníkem, které se mají do archivu přidat.

ArchiveExtractorSettings

Zde jsou načteny přípony souborů, které se extrahují a které se ignorují. Přípony jsou psány do jednoho řetězce a jsou odděleny čárkou.

- SupportedExtensions: přípony archivů, které se extrahují.
- IgnoredExtensions: přípony souborů, které se mají ignorovat.

5.2 OutputWritter

Tento modul je využíván všemi ostatními moduly k vypisování informací do konzole a do logovacích souborů. Obsahuje pouze jednu třídu *OutputInformationWritter*, které je předána instance na logovací soubor. Standardně se při vkládání závislostí vytvoří nová instance dané třídy. V tomto případě to však není žádoucí, neboť všechny moduly a třídy musí zapisovat do stejného logovacího souboru. Při spuštění jak webové, tak konzolové aplikace se vytvoří instance *LogManager* z knihovny *log4net*, která vytvoří nový logovací soubor s názvem:

```
AppLog_yyyy_mm_dd_hh_dd_mm.log
```

Jednotlivá písmena ve výše uvedeném názvu postupně značí: rok, měsíc, den, hodinu, minutu a sekundu vytvoření souboru. Jméno souboru se načítá z nastavení aplikace, které se nachází v souboru *app.config*. Instance tohoto logovacího souboru je poté předána do všech modulů, kde je potřebná. Zápisy do logovacího souboru lze rozdělit podle závažnosti. K tomu slouží výčtový typ *LoggerLevels*, který nabízí možnosti: Info, Debug, Warn, Error a Fatal. V této aplikaci jsou využívány 3 úrovně. Info je využíváno pro zapisování informací o procesu importu a všeho, co s tím souvisí. Zapisuje se tedy např. jméno uživatele, který práci vykonává, metody, které jsou volány apod. Dále je použita úroveň Error, která slouží k zápisu chyb, které se týkají samotného vykonávání procesu importu. Tedy např. když nelze extrahovat archiv, protože byl poškozen, pokud nelze nalézt archivy s MIB soubory v nástroji JIRA apod. Nejsou však zde zapisovány chyby, které ohrožují chod celého programu, ty se zapisují s úrovni Fatal.

Při vývoji bylo nutné použít starší verzi knihovny *log4net* pro práci s logovacími soubory, a to verzi 1.2.10 namísto nejnovější verze 2.0.1. A to proto, že MIB kompilátor taktéž využívá knihovnu *log4net*, avšak právě starší verzi 1.2.10. MIB kompilátor je totiž poněkud starší, avšak dobře fungující program, který jsem měl dostupný pouze jako knihovnu s příponou .dll. Kvůli tomu jsem nebyl schopen změnit verzi knihovny *log4net* na novější. V případě importování MIB souborů je však verze 1.2.10 zcela dostačující.

5.3 CommonTasks

V tomto modulu se nachází třídy s algoritmy, které se využívají na více místech této aplikace. Proto jsem se rozhodl je vložit do jednoho modulu, aby nedocházelo ke zbytečné duplicitě kódu. Jedná se především o práci se soubory a adresáři. Jsou zde celkem tři třídy. První třída s názvem *DirectoryAndFileHelper* slouží k práci s adresáři a soubory. Jsou zde tedy metody pro získání všech adresářů a souborů v adresáři, pro získání jména adresáře, jména souboru bez přípony apod. Další třída *ItemTypeProvider* vrací typ položky, který je jí předán, tedy zda se jedná o adresář, podporovaný archiv, či soubor, který se má ignorovat. Poslední třída *ProgramInstallationChecker* zjišťuje, zda je program, jehož jméno je předáno, nainstalován na počítači. K tomu je použita třída *RegistryKey* ze systémových knihoven, která zapouzdřuje práci nad registry operačního systému Windows. Je zde tedy předán název programu a následně jsou prohledány 32-bitové a 64-bitové registry. Pokud je v registrech nalezen záznam s jménem odpovídající hledanému programu, je vrácena celá cesta k tomuto programu. Tímto způsobem může být nalezen jakýkoli program pouze podle jeho jména a nezáleží, na kterém disku a ve kterém adresáři je nainstalován. Díky tomu nebylo nutné zadávat do nastavení aplikace další položky pro cestu k programům *7zip* a *Notepad++*. Program si je jednoduše nalezne sám.

5.4 ArchiveExtractor

Tento modul slouží k extrahování archivů s MIB soubory, které se stáhnou ze sdíleného úložiště firmy SolarWinds. Zákazníci soubory posílají v archivech libovolné struktury a zanoření s množstvím dalších druhů souborů, jako např. pdf, txt, doc a podobně. Je tedy potřeba extrahovat všechny archivy, které se v adresáři daného JIRA úkolu týkají. Jak je zmíněno výše, firma SolarWinds již měla takový nástroj, který se však zasekával, nepodporoval všechny typy archivů a také nedokázal procházet libovolnými úrovněmi zanoření adresářů. Bylo tedy zapotřebí vytvoření nového nástroje, který všechny uvedené požadavky bude splňovat.

Pro extrahování byl vybrán program *7zip*. Nabízelo se také použití některých dostupných knihoven jako např. *SevenZipSharp* nebo *SevenZipExtractor*. Žádná dostupná knihovna však nenabízela podporu pro všechny typy archivů, kterou nabízí samotný program *7zip*. Bylo by nejspíše potřeba využít více knihoven najednou a podle přípony archivu zvolit použití vhodné knihovny. Vytvářela by se tak prakticky nástavba na knihovny, které na pozadí používají program *7zip*.

K extrahování je tedy potřeba mít na počítači nainstalovaný program *7zip*. Kontrola, zda tomu tak skutečně je, probíhá před samotným spuštěním extrahování archivů. K tomuto zjištění je použita třída *ProgramInstallationChecker* z modulu *CommonTasks*, který je popsán v podkapitole 5.3. Pokud program *7zip* není nainstalován, vypíše se tato událost uživateli a import je ukončen.

V případě, že program *7zip* nainstalován je, dojde k zaznamenání všech souborů a adresářů v zadaném adresáři na nejvyšší úrovni bez zanoření. Všechny položky jsou přidány do seznamu, přes který se následně iteruje. Při celém procesu se uchovávají informace o extrahovaných a neznámých souborech, pravdivostní hodnota o úspěchu, případné chybové zprávy při neúspěchu a seznam souborů, které nebylo možné extrahovat a z jakéhokoli důvodu. Tato data jsou pak zobrazována uživateli. Následně se v cyklu proiterují všechny položky v adresáři a uloží se do odpovídajích seznamů. Na základě typu položky se provede odpovídající akce:

- Adresář: Pokud se jedná o adresář, načtou se opět všechny položky v adresáři bez jakékoli úrovně zanoření a následně se roztřídí. Pokud se jedná o soubor s neznámou příponou, je tento soubor přidán mezi neznámé soubory a dále se s ním nepracuje. Pokud se jedná o adresář nebo podporovaný archiv, je tento soubor/adresář přidán do seznamu položek, které se mají zpracovat.
- Ignorovaný soubor: Ignorovaný soubor není potřeba nijak zpracovávat a přejde se tedy na další soubor.
- Neznámý soubor: Neznámý soubor je pouze přidán do seznamu neznámých souborů.
- Podporovaný archiv: Provede se extrahování daného archivu 5.4.1.

Ať se jedná o jakýkoli typ souboru, či o adresář, je položka vždy vymazána ze seznamu s položkami určenými ke zpracování, aby nedocházelo k opakovanému zpracovávání položek, a tudíž k zacyklení.

5.4.1 Extrahování archivu

Nejprve se zjistí, zda je archiv extrahovatelný. K tomu je taktéž využito programu *7zip*. Ve třídě *CommandLineExtractor* je spuštěn nový proces, který spustí příkazový řádek na pozadí s následujícím příkazem:

```
7z t -p archiv >nul 2>nul
```

7z reprezentuje celou cestu k programu *7zip*. Přepínač *t* slouží ke zjištění integrity archivu. Přepínač *-p* slouží ke zjištění, zda archiv není zaheslován. Dále pak dochází k přesměrování standardního výstupu a standardního chybového výstupu pomocí přepínačů *>null* a *2>null*. Po spuštění tohoto příkazu se čeká na návratovou hodnotu. Pokud je návratová hodnota rovna 2, je soubor buď zaheslovaný, nebo poškozený. Ani v jednom případě tak nelze soubor extrahovat a soubor je přidán do seznamu souborů, které nelze extrahovat. Jelikož by mohlo dojít k zamrznutí procesu nebo jeho pozdržení, je tento úkol časově ohraničen. Pokud doba zjišťování, zda je archiv extrahovatelný, přesáhne 5 sekund, je archiv rovněž přidán do seznamu s neextrahovatelnými soubory. Takto nedojde k zamrznutí celého programu. Uživatel se však dozví o těchto souborech a může se je pokusit ještě ručně zpracovat.

Pokud je archiv extrahovatelný, uloží se do seznamu všechny položky, které se v adresáři s archivem nacházejí. Poté se provede samotná extrakce. Jestliže se provede úspěšně, znovu se načtou všechny položky z daného adresáře a do seznamu s položkami ke zpracování se přidá rozdíl listů vytvořených před a po extrahování. Takto se do seznamu s položkami ke zpracování nepřidají položky, které tam již jsou, ale pouze ty, které se objevily po extrahování.

Samotné extrahování se provádí ve třídě *CommandLineExtractor* a stejně jako u zjišťování, zda je archiv extrahovatelný, se spouští příkazový řádek na pozadí. K extrahování byl využit následující příkaz:

```
7z x archiv -o cílovy_adresář -aoa
```

7z reprezentuje celou cestu k programu *7zip*. Přepínač *x* slouží k extrahování archivu, který je zadán svou úplnou cestou. Dále přepínač *-o* určuje cílový adresář, kde se má archiv extrahovat. Píše se přímo za tento přepínač. Poslední přepínač nastavuje přepisovací pravidla, která jsou takto nastavena na přepsání (určuje poslední znak *a* v tomto přepínači).

Při tomto procesu bylo také nutné zjišťovat, zda proces nezamrznul. Některé archivy mají velkou velikost a jejich extrahování tak trvá dlouhou dobu. Nelze tedy použít pouze časový limit jako u zjišťování extrahovatelnosti archivu. *7zip* v příkazovém řádku však nenabízí možnost průběžného informování o stavu extrahování. Zvolil jsem tedy variantu průběžného sledování adresáře, do kterého se archiv extrahuje. Po spuštění příkazu pro extrahování probíhá periodické dotazování na adresář, zda v něm došlo k nějké změně. Jedná se o následující změny: čas posledního přístupu, čas posledního zápisu a velikost adresáře. Pokud se změní kterákoli z těchto hodnot v době 3 sekund, extrahování stále pokračuje. V opačném případě se sledování ukončí a počká se 1 sekundu na konec extrahování. Takto lze dostat chybovou zprávu s návratovou hodnotou, kterou je možné předat uživateli.

Program *7zip* dokáže extrahovat prakticky kterýkoli typ komprese, což je jeho velká výhoda. Má však ještě jednu výhodu, která je u tohoto projektu cenná. Program *7zip* nejprve zvolí algoritmus pro dekompresi podle přípony archivu. Pokud se však archiv nepodaří extrahovat, načte si několik prvních bytů, které jednoznačně určí, jaký kompresní algoritmus

byl použít, a k tomu zvolí adekvátní dekompresní algoritmus. Zákazníci firmy SolarWinds bohužel někdy zašlou archivy, u kterých přípona archivu nesouhlasí s kompresním algoritmem. Díky tomu, že program *7zip* tuto situaci odhalí, není nutné JIRA úkoly zavírat s chybou, že archivy nelze extrahovat. Pouze je vypsána upozorňující zpráva, že se nepodařilo archiv extrahovat jako typ archivu odpovídající koncovce archivu, ale podařilo se archiv extrahovat algoritmem odpovídající prvním několika bytům.

5.5 JiraCommunicator

Tento modul slouží ke komunikaci s nástrojem JIRA. Jak je napsáno v kapitole 3.2.1, tento modul již byl z části vytvořen a umožňoval přihlášení do nástroje JIRA, stažení všech relevantních úkolů, a nakonec také stažení samotných MIB souborů ze sdíleného úložiště firmy SolarWinds. Všechny tyto činnosti byly využity i při automatizaci importu MIB souborů. Výše uvedené činnosti však byly v hlavní metodě *Main* tohoto programu. Bylo proto nutné tento velký blok kódu rozdělit do menších částí a do samostatných metod, které bude možné nezávisle volat. Vše ohledně importu MIB souborů je navázáno na JIRA úkoly. V tomto modulu se nachází třída *JiraIssue*, která obsahovala následující informace o daném úkolu:

- IssueId: identifikátor úkolu.
- NetSuiteId: identifikátor úkolu v nástroji NetSuite. Na základě úkolů, které zadá zákazník do NetSuite, vytvoří zaměstnanci firmy SolarWinds z oddělení pro podporu zákazníků úkoly v nástroji JIRA.
- Status: informuje o stavu, ve kterém se úkol nachází. Může se nacházet ve 3 stavech: *ToDo* - úkol určen ke zpracování, *In progress* - úkol je zpracováván, *Closed* - úkol je uzavřen.

IssueId a *NetSuiteId* dále slouží k pojmenování adresářů, do kterých se MIB soubory stahují. Díky těmto identifikátorům je možné na serveru, kde import probíhá, MIB soubory snadno zpětně dohledat. Dále zde byla metoda *StartInvestigation*, která na začátku stahování MIB souborů změnila stav úkolu z *ToDo* na *In progress*, čímž se úkol přesunul do sloupce *In progress* také v nástroji JIRA. V popisu úkolu se rovněž objeví jméno zaměstnance, který úkol do tohoto stavu přesunul. Jelikož tento program nebyl původně určen k celému importu MIB souborů, bylo do třídy *JiraIssue* nutné přidat další položky, které budou uchovávat potřebná data. Přidány byly tyto položky:

- SalesForceId: identifikátor úkolu v nástroji SalesForce.
- Downloaded: pravdivostní hodnota informující, zda se podařilo stáhnout MIB soubory ze sdíleného uložiště.
- DownloadedError: chybová hláška o stahování MIB souborů ze sdíleného úložiště v případě, že se stahování nezdaří.
- ExtractorResult: výsledek extrahování archivů daného JIRA úkolu. Jedná se instanci třídy *EtractorResults*, která obsahuje informace o extrahovaných archivech, neextrahovaných archivech, neznámých souborech, pravdivostní hodnotě o úspěchu extrahování a případné chybové výstupy.

- CompilerSuccess: pravdivostní hodnota informující, zda se podařilo zkompilovat MIB soubory daného JIRA úkolu.
- CompilerError: chybová hláška o kompilaci MIB souborů v případě, že došlo k chybě, po které není možné dále MIB soubory kompilovat.
- CompilerErrorList: seznam chyb v MIB souborech daného JIRA úkolu.
- IOException: chybová hláška informující o neočekávané chybě v průběhu celého importu MIB souborů, kvůli které nelze pokračovat v importu daného JIRA úkolu.

Do této třídy jsou tedy přidány položky pro každý krok v celém procesu importu MIB souborů. Takto jsou všechna data pohromadě a je snazší je předat uživateli. Během vývoje došlo ve firmě ke změně používaného nástroje v oddělení pro podporu zákazníků. Firma přešla z nástroje *NetSuite* na nástroj *SalesForce*. Tímto se změnilo označování JIRA úkolů. Starší úkoly ještě mají *NetSuiteId*, novější však už jen *SalesForceId*. Je zde tedy dodána ještě položka *CaseId*, která zastřešuje obě zmíněné možnosti a navrací vždy Id, které je u daného JIRA úkolu aktuální. Bylo však potřeba změnit řetězec *Jql* pro filtrování JIRA úkolů. Níže je k vidění nová verze:

```
project = CUST AND (comment = "Area: MIB Submission"OR "Product
Category-"MIB Submission") AND (Product = "Network Performance Monitor"OR
 Product = "Network Configuration Manager"OR Product = "NetFlow Traffic
  Analyzer") AND (Type = "Customer Feature Request"OR Type = "Customer
  Issue") AND (status != Closed OR updated > -8d) OR (Type = "Customer
Feature Request"OR Type = "Customer Issue") AND ("Product Version-NPM OR
"Product Version-NCM OR "Product Version-NTA OR "Product Version-"Network
performance monitor") AND "Product Category"- "MIB Submission"AND (Status
                      != Closed OR updated > -8d)
```

Oproti verzi uvedené v kapitole 5.1 zde přibyla celá větev, která představuje druhou alternativu zahrnující použití *SalesForceId*. Tato větev vyžaduje, aby úkol byl typu *Customer Feature Request* nebo *Customer Issue*. Zároveň se musí jednat o spojení s produktem NPM (Network Performance Monitor), NCM (Network Configration Manager), nebo s NTA (Network Traffic Analyzer). Kategorie produktu musí být *MIB Submission* a úkol nesmí být ve stavu *Closed* a poslední změna nesmí být starší než 8 dní. Tato větev je tedy velice podobná té první.

Pro využití *Dependency injection* bylo nutné pouze extrahovat rozhraní ze stávajících tříd. Po těchto úpravách bylo možné modul v automatizaci použít.

5.6 SupportSteps

Tento modul poskytuje implementace ke krokům, které následují po importu MIB souborů do MIB databáze. Provádějí se až po úplném ukončení importu všech JIRA úkolů. Kroky zálohování databáze, vytvoření archivu databáze a nahrání databáze se provádí po zpracování všech JIRA úkolů, a to i v případě, pokud se nepodařilo úspěšně všechny MIB soubory importovat. Vytvoření nové verze MIB databáze nemůže být odloženo pokud někteří zákazníci například nepřiloží své MIB soubory k úkolu, nebo jsou soubory poškozeny. Tyto úkoly se zavřou s příslušnou chybovou zprávou a zákazníci mají možnost znovu soubory dodat

nebo vytvořit nový úkol, který bude zpracován při dalším importu MIB souborů. Všechny třídy, které vytváří nové procesy na pozadí užívají příkaz *using*. V případě jakékoli chyby tak automaticky dojde k uvolnění všech zdrojů, které proces používá. Dále tento modul také nabízí správu nastavení celé aplikace odeslání emailu po dokončení celého procesu importu MIB souborů.

Nastavení aplikace

Provádí jej třída *AppConfigModifier*, která umožňuje načíst veškeré nastavení, ale také jej uložit v případě jakékoliv změny. Tato možnost je dostupná pouze z webové aplikace. Všechny položky nastavení jsou poté uchovávány ve třídě *AppConfiguration*, která je předávána webové aplikaci a případně z ní také přijímána. K načítání a ukládání je využita třída *ConfigurationManager*. Pro její použití je nutné do projektu přidat knihovnu se stejným jménem[1], která umožňuje práci s konfiguračními soubory aplikace. Dojde tedy k otevření konfiguračního souboru *app.config* spustitelného projektu. Tento soubor je psán ve formátu *xml*. Jsou zde dvě hlavní sekce nastavení: *appSettings* a *connectionStrings*. První z nich je určena pro nastavení aplikace, druhá pro připojovací řetězce např. k databázi jako v případě tohoto projektu. Všechny hodnoty jsou uloženy jako dvojice s klíčem a hodnotou. Na základě daného klíče je tedy možné hodnotu načíst, nebo uložit. Je však potřeba rozlišovat, zda načítáme z *appSettings* či *connectionStrings*. V případě konzolové aplikace je nutné nastavení měnit rucně v souboru *app.config*.

Zálohování databáze

Než se vytvoří nový archiv MIB databáze, je potřeba zálohovat ten stávající. Pokud by se z jakéhokoli důvodu stala nová databáze nepoužitelnou, nebo by obsahovala chybu, je nutné mít k dispozici poslední funkční verzi databáze. K zálohování databáze slouží třída *BackUpDb* v modulu *SupportSteps*. V této třídě se vytvoří nový proces, který spustí příkazový řádek na pozadí. V něm je pak spuštěn soubor *backupfile.bat*. Jelikož soubor *backupfile.bat* spouští další skript *backupfile.ps1*, který se spouští v programu *PowerShell*, je nutné, abychom při spuštění tohoto skriptu byli v adresáři, ve kterém se nachází uvedený skript. Příkazy jsou tak v procesu spuštěny dva. První pro vstup do odpovídajícího adresáře a druhý pro spuštění souboru *backupfile.bat*. Kvůli spuštění skriptu v programu *PowerShell* nebylo možné získat standardní výstup a standardní chybový výstup ze spuštěného procesu jako např. u extrahování archivů. Na standardní výstup a standardní chybový výstup tedy byly zaregistrovány události, které se o jejich čtení starají. Díky tomu je možné uživateli předat informace o stavu záhohování databáze.

Vytvoření archivu databáze

Komprese MIB databáze se provádí zejména z důvodu šetření místa na serveru, kde se zpracovávání MIB souborů provadí. Samotná databáze zabírá již přes 1,7 GB místa na disku, kdežto zkomprimovaná databáze zabírá okolo 300 MB. Vytváření archivu databáze se provádí ve třídě *ArchiveCreator*. Vytváření archivu je velice podobné extrahování. Taktéž se na pozadí spustí příkazová řádka v novém procesu. Ke kompresi je taktéž použit program *7zip*. Níže lze vidět používaný příkaz:

[1] https://www.nuget.org/packages/System.Configuration.ConfigurationManager/4.4.1

```
7z a jméno_archivu soubor1 soubor2
```

Příkaz *7z* reprezentuje úplou cestu k tomuto programu. Parametr *a* slouží pro přidání souborů do archivu. Poté následuje úplné jméno archivu včetně úplné cesty, kde má být vytvořen. Nakonec jsou zde dva soubory, které se do archivu přidávají. Jedná se o samotnou MIB databázi a soubor *readme-mm-dd-yyyy.pdf* obsahující informace o používání databáze, kde jednotlivá písmena postupně značí měsíc, den a rok. Jméno archivu včetně jeho cesty je načteno z konfiguračního souboru. V adresáři, kde se nachází databáze, se nachází také uvedený soubor *readme*. Tento soubor je nutné přejmenovat. V jeho jméně je datum poslední změny databáze. Je tedy zapotřebí toto datum aktualizovat. Nejprve je nutno tento soubor nalézt v adresáři, neboť nemůže být uveden v konfiguračním souboru, jelikož se jeho jméno mění při každém importu MIB souborů. V daném adresáři jsou tak nalezeny všechny soubory s příponou **.pdf*. Dále pomocí níže uvedeného regulárního výrazu hledáme soubor s odpovídajícím jménem.

```
readme-[0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9].pdf
```

Pokud žádný soubor není nalezen, je vytváření archivu ukončeno s hláškou, že se soubor *readme* nepodařilo najít. V opačném případě je brán první soubor, který odpovídá tomuto regulárnímu výrazu. I kdyby se v adresáři nacházelo více souborů, které by odpovídaly tomuto regulárnímu výrazu, nezáleží na tom, který se do archivu přidá. Všechny soubory by totiž měly stejný obsah. Mění se pouze jejich jméno.

Bohužel se mi nepodařilo najít způsob, jak zjistit, jestli vytváření archivu stále probíhá nebo došlo k zamrznutí procesu. Bylo tedy zvoleno provizorní řešení, kdy je kontrolována doba vytváření archivu databáze. Běžně vytváření archivu zabere okolo 5 minut. Pro tento proces je zvolen časový limit 20 minut, lze jej však konfigurovat v nastavení aplikace. Pokud do jeho vypršení proces neskončí, je ukončen s chybovou zprávou, že se archiv nepodařilo vytvořit.

Nahrání databáze na server

Po vytvoření archivu nové verze MIB databáze je nutné ji nahrát na server, odkud si ji mohou stáhnout zákazníci firmy SolarWinds. Nahrání databáze se provádí pomocí aplikace *PurgeApp*, kterou již firma měla k dispozici. Jedná se o spustitelný soubor s názvem *PurgeApp.exe*. Volání tohoto programu se provádí stejně jako v případě zálohování databáze. Na pozadí taktéž spustí příkazová řádka v novém procesu. Zde se již jen vstoupí pomocí příkazu *cd* do adresáře, který se načte z konfiguračního souboru aplikace a spustí se příslušný soubor, jehož jméno je taktéž uvedeno v konfiguračním souboru. Spouštění tohoto programu je implementováno ve třídě *PurgeAppCaller* v modulu *SupportSteps*. Ověření, zda se podařilo databázi na server nahrát, se prodvádí pomocí návratové hodnoty aplikace *PurgeApp*. Pokud skončí s kódem 0, vše proběhlo v pořádku. Pokud ne, je načten také chybový výstup a ten je poté předán uživateli.

Odeslání emailu

Jak je zmíněno v kapitole 4.1, po ukončení importu MIB souborů se odesílá email manažerovi týmu s informacemi o jeho průběhu. Odesílání emailu je naprogramováno ve třídě

MailSender v modulu *SupportSteps*. Je zde využito třídy *SmtpClient*, která umožňuje zaslání emailu pomocí protokolu SMTP[2]. Při vytváření instance uvedené třídy je nutné ji předat jako parametr adresu SMTP serveru, ze kterého se email bude odesílat. Kromě emailové adresy odesílatele a hesla k jeho emailové schránce jsou všechny ostatní důležité položky načteny z konfigurace aplikace. Tyto položky jsou popsány v kapitole 5.1. Emailovou adresu odesílatele a odpovídající heslo zadá uživatel ve webovém rozhraní, případně v konzolové aplikaci. Konzolová aplikace vytáhne všechna potřebná data z konfigurace, tedy předmět emailu, emailovou adresu příjemce a tělo emailu. V konfiguraci aplikace jsou však těla emailu dvě. Jedno pro případ úspěšného importu MIB souborů a druhé pro opačnou situaci. Na základě výsledku celého procesu je tedy odeslán odpovídající email. V konzolové aplikaci však není možné těla emailu změnit. Webová aplikace toto umožňuje. Z nastavení aplikace jsou načteny obě varianty těla emailu. Uživatel si zvolí, kterou šablonu chce použít. Dále ji může libovolně upravovat. Při stisknutí tlačítka pro odeslání emailu se upravené tělo emailu odešle třídě *MailSender* a ta jej použije namísto šablony z konfigurace aplikace. Nakonec už je potřeba pouze nastavit port na číslo 25, které protokol SMTP používá a určit kódování textu. V této aplikaci je použito kódování UTF8[3].

5.7 JiraCasesProcessor

Tento modul řídí celý proces importu MIB souborů a má reference na všechny výše uvedené moduly. Jsou zde 4 hlavní třídy, které se starají o průběh celého procesu. Níže jsou tyto třídy blíže popsány.

5.7.1 Řízení celého procesu

Probíhá v hlavní třídě s názvem *ProcessControl*, ze které je možné volat všechny ostatní třídy, ale také moduly v celé aplikaci, a také uchovává informace o všech JIRA úkolech v kolekci s položkami typu *IIssue* (rozhraní, které implementuje třída *JiraIssue* popsaná v kapitole 5.5). Instance třídy *ProcessControl* tedy musí být stále dostupná webovému rozhraní. Instance této třídy je vytvořena ve třídě *ServiceResolver* za pomocí *Dependency injection*, a to jako *Jedináček*. Je tedy možné vytvořit pouze jedinou instanci dané třídy a ta je vkládána všude, kde je potřebná. Tato funkcionalita však není ošetřena přímo v kódu dané třídy, ale provádí ji za nás *Dependency injection*. Tato třída se používá ve webovém rozhraní pro správu JIRA úkolů, neboť pokaždé, kdy je vyslán dotaz z webových stránek, musí být k dispozici stejná instance se všemi JIRA úkoly. Takto se projeví všechny změny provedené na JIRA úkolech. Tato třída tedy obsahuje metody pro spouštění všech kroků pro import MIB databáze.

5.7.2 Zpracování JIRA úkolů

Zpracování JIRA úkolů provadí třídy *AllJiraCasesProcessor* a *SingleJiraCasesProcessor*. Při spuštění konzolové aplikace dochází automaticky ke zpracovávání všech JIRA úkolů. Je tedy volána *AllJiraCasesProcessor*, která tuto akci řídí. Seznam všech JIRA úkolů je postupně procházen a jednotlivé úkoly jsou zpracovávány pomocí třídy *SingleJiraCasesProcessor*. Po zpracování každého úkolu dojde k ověření, zda se jeho import zdařil, či nikoli. V obou případech je výsledek zaznamenán do logovacího souboru a také předán uživateli.

[2] https://tools.ietf.org/html/rfc821
[3] http://www.utf-8.com/

Zpracování všech JIRA úkolů je možné spustit také pomocí příslušného tlačítka na webovém rozhraní.

Zpracování jediného úkolu provádí metoda *ProcessJiraCase* ve třídě *SingleJiraCaseProcessor*. Této metodě jsou předány následující parametry: instance daného úkolu, aktuální datum, úplná cesta k programu *Notepad++* a pravdivostní hodnota určující, zda se jedná o volání z konzolové či webové aplikace. Zpracování úkolu se skládá ze tří kroků: stažení archivů s MIB soubory ze sdíleného úložiště, extrahování těchto archivů a nakonec kompilace MIB souborů. Výsledek každého kroku se nahraje do instance daného úkolu. Pokud v některém z kroků dojde k chybě a nepodaří se jej úspěšně provést, je zpracování daného úkolu ukončeno a chybová zpráva je zobrazena uživateli. Parametr s aktuálním datem je používán při stahování archivů s MIB soubory, který rovněž tyto archivy kopíruje do pracovního adresáře. Úplná cesta k programu *Notepad++* je používána při importování MIB souborů, kdy jsou poškozené soubory otevřeny v tomto programu na řádce, na které se chyba nachází. Pokud se jedná o konzolovou aplikaci, program se při kompilaci MIB souborů dotazuje uživatele, zda jsou soubory opraveny, či nikoli. Pokud ano, je kompilace spuštěna znovu. V opačném případě je kompilace ukončena. Ve webové aplikaci se po opravě chyb v souborech spustí kompilace znovu příslušným tlačítkem. Jednotlivé kroky zpracování MIB souborů lze z webového prostředí spouštět také samostatně. Pokud by tak došlo k neočekávané chybě, kterou však lze ručně opravit, nebylo by nutné znovu provádět všechny kroky.

Při kompilování MIB souborů se využívá knihoven z programu *CompilerCore* vytvořeného firmou SolarWinds. Při automatizaci importu je využívána pouze knihovna tohoto programu. Projekt tak není celý připojen, pouze je přidán jako reference k modulu *JiraCasesProcessor*. Volání kompilátoru je implementováno ve třídě *Compiler*, která je volána ze třídy *SingleJiraCaseProcessor*. Ten pro svou činnost potřebuje zejména řetězec pro připojení k MIB databázi, který je načten z konfigurace aplikace. Z webového rozhraní je možné jej měnit. Kompilátor se spouští nad adresářem a pokouší se kompilovat a importovat všechny soubory, které v názvu mají řetězec *mib*, či *MIB*. Ostatní soubory jsou ignorovány. Kompilátor jako výsledek vrací instanci třídy *CompilerResult*, která obsahuje pravdivostní proměnnou informující o výsledku importu, list se soubory importovanými do databáze, list se soubory, které se nepodařilo importovat, a nakonec list s kompilačními chybami. Pokud lze MIB soubor zkompilovat, je přidán do databáze a také do listu importovaných souborů. V opačném případě je soubor přidán do listu neimportovaných souborů a chyba je zaznamenána. Kompilátor se pokusí zkompilovat všechny soubory v adresáři. Pokud jsou např. v prvním kompilovaném souboru chyby, kompilace se neukončí. Chyby se pouze zaznamenají a pokračuje se na další soubor. Jelikož se kompilují najednou vždy všechny MIB soubory v adresáři, dochází často k opětovnému importu MIB souborů, které již byly importovány. Kompilátor se pokusí importovat každý soubor z adresáře bez ohledu na to, zda-li byl již importován, či nikoli. Po importu je procházen jeho výsledek. Pokud jsou zde chyby, jsou chybné soubory otevřeny v programu *Notepad++*.

5.8 Webové rozhraní

Původně byl na webové rozhraní využit framework *ASP.NET Core*[4] ve verzi 2.0. Jedná se o nejnovější framework společnosti *Microsoft*[5]. Všechny moduly však byly vyvíjeny ve verzi

[4] https://docs.microsoft.com/cs-cz/aspnet/core/

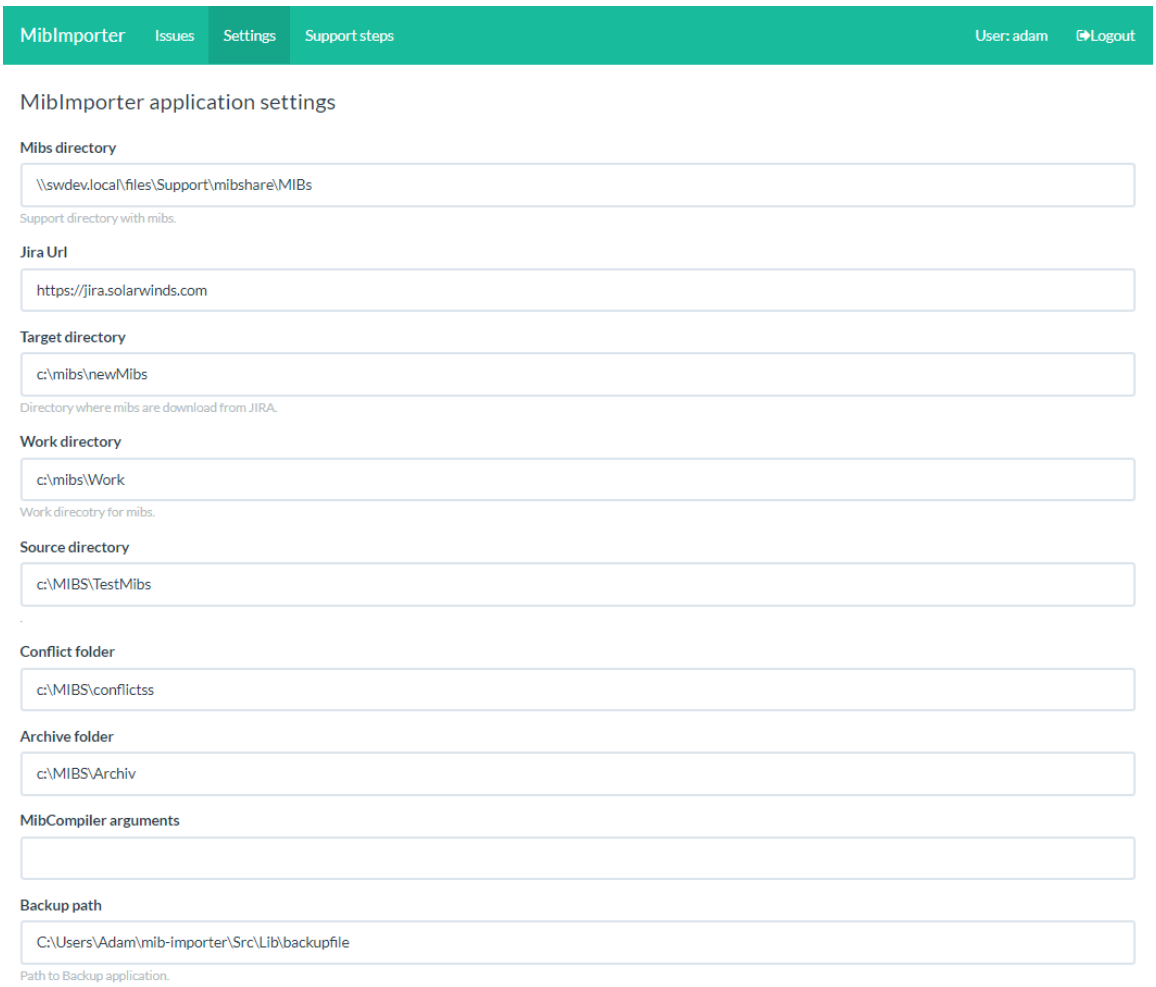
[5] https://www.microsoft.com/cs-cz

.NET 4.6.1, se kterou však *ASP.NET Core* není kompatibilní. Není zde možné používat všechny knihovny, které jsou v modulech popsaných výše. Webové rozhraní tak bylo vytvořeno pomocí *.NET Framework 4.6.1*. Frontend využívá framework *Angular CLI*[6] ve verzi 1.7.4. Pro webové rozhraní byl vytvořen zcela nový projekt s názvem *MibImporterWebApplication*. Do tohoto projektu byly přidány všechny existující moduly reprezentující jednotlivé moduly, které byly popsány výše. Takto se veškeré změny provedené v těchto modulech projeví také v konzolové aplikaci, která tyto moduly využívá. Ke komunikaci webového rozhraní s kontroléry byl použit REST, kde byly využívány metody typu *Get* a *Post*. Komunikaci lze rozdělit do čtyř druhů. Prvním z nich je autorizace uživatele, následuje práce s nastavením aplikace, podpůrné kroky z kapitoly 5.6 a nakonec komunikace ohledně zpracování MIB souborů. Tuto komunikaci provádějí kontroléry, které jsou volány z webového rozhraní. Cesta pro jejich volání je upřesněna pomocí třídního atributu *Route*. Každá metoda má rovněž svůj textový identifikátor pro volání správných metod. Webová aplikace obsahuje 3 záložky pro práci, které se nachází v panelu v horní části stránky. Jedná se o záložky *Issues*, *Settings* a *Support steps*. Pokud není uživatel přihlášen, jsou mu tyto záložky skryty. Jelikož webový projekt používá svůj vlastní *Dependency injection*, bylo do něj nutné přidat knihovnu *Castle.Windsor.MsDependencyInjection*[7], která umožňuje použití *CastleWindsor* i v tomto webovém projektu. U všech procesů na webovém rozhraní, které mohou trvat delší dobu, se po jejich spuštění zobrazí ikona točícího kolečka. Celá stránka zešedne a je nastavena do režimu *diseabled*, kdy není možné spustit další akce. Po dokončení procesu se stránka vrátí do původního stavu. Oproti návrhu webového rozhraní v kapitole 4 došlo ke změně grafiky. Logika importování MIB souborů však zůstala zachována. Byly navíc přidány stránky s detailem JIRA úkolu a s pomocnými nástroji.

Autorizace je prováděna v kontroléru *AuthController* metodou *Login*, která je typu *Post*. Do této metody jsou zaslány přihlašovací údaje do nástroje JIRA. S těmito údaji je následně volána metoda ze třídy *AllJiraCaseProcessor* pro přihlášení do nástroje JIRA. Pokud přihlášení proběhne v pořádku, je uživateli vygenerován token, který je následně předáván při jakékoli akci na webovém rozhraní. Tento token je generován pomocí knihovny *Authentication.JwtBearer*[8]. Pro provádění všech akcí musí být uživatel přihlášen. Aby se neautorizovaný uživatel nedostal na jinou stránku než úvodní s přihlašovacím formulářem a nemohl využívat metody z ostatních kontrolérů, je ve všech kontrolérech přidán atribut *Authorize*.

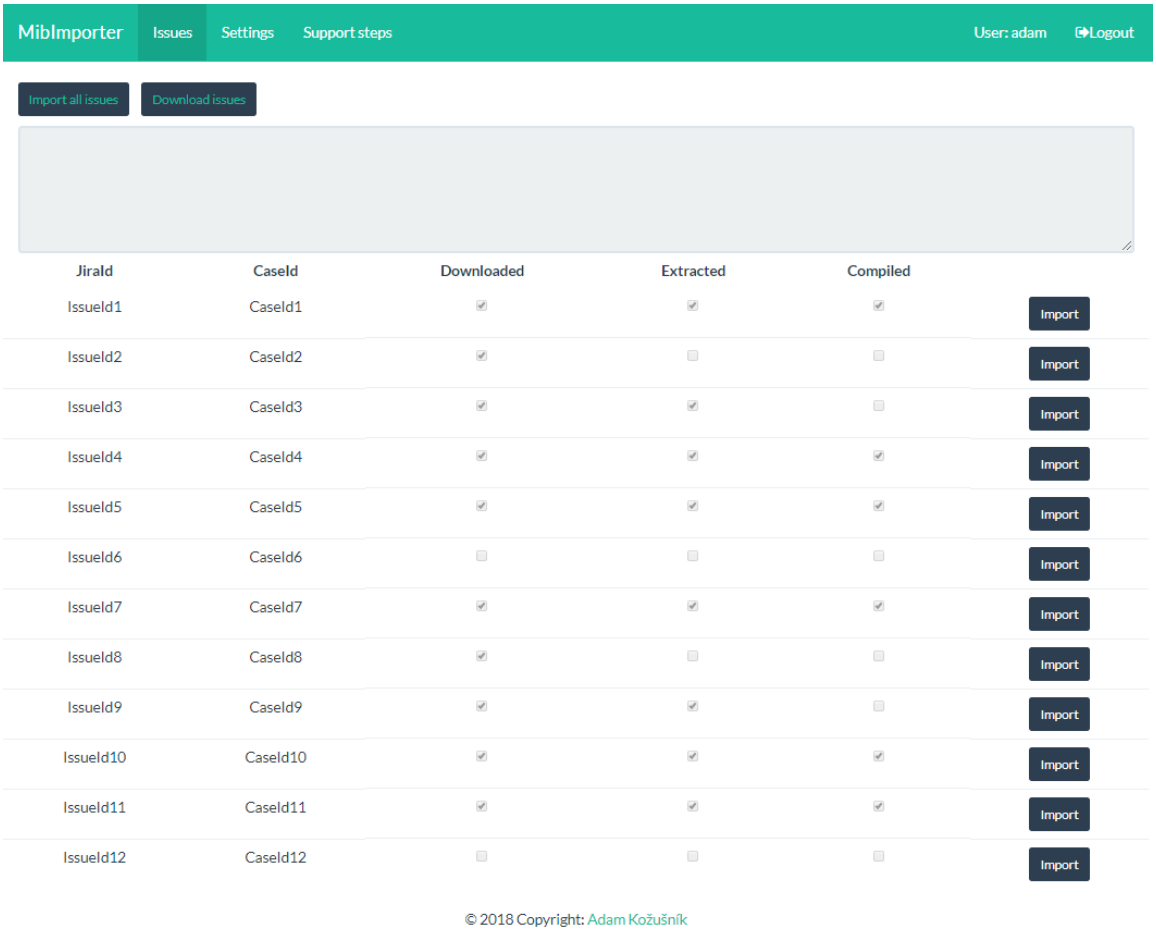
[6] https://cli.angular.io/
[7] https://www.nuget.org/packages/Castle.Windsor.MsDependencyInjection/
[8] https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.JwtBearer/

Nastavení aplikace lze upravovat v záložce *Settings* nacházející se v panelu umístěném v horní části webové stránky. Stránku s nastavením lze vidět na obrázku 5.1. Všechny parametry, které lze nastavit se nacházejí ve formuláři a jsou umístěny pod sebou. Pro zobrazování těchto parametrů je použita komponenta *input*. Pro parametry s delším textem pak komponenta *textarea* umožňující zobrazovat text ve více řádcích. U jednotlivých parametrů jsou také popisky, které upřesňují účel parametru. Na konci stránky se nachází dvě tlačítka. První tlačítko *Reload* slouží ke znovunačtení všech parametrů. Druhé tlačítko *Save* slouží k uložení zadaných parametrů. Zpracování požadavků z této stránky je implementováno v kontroléru *SettingsController*.



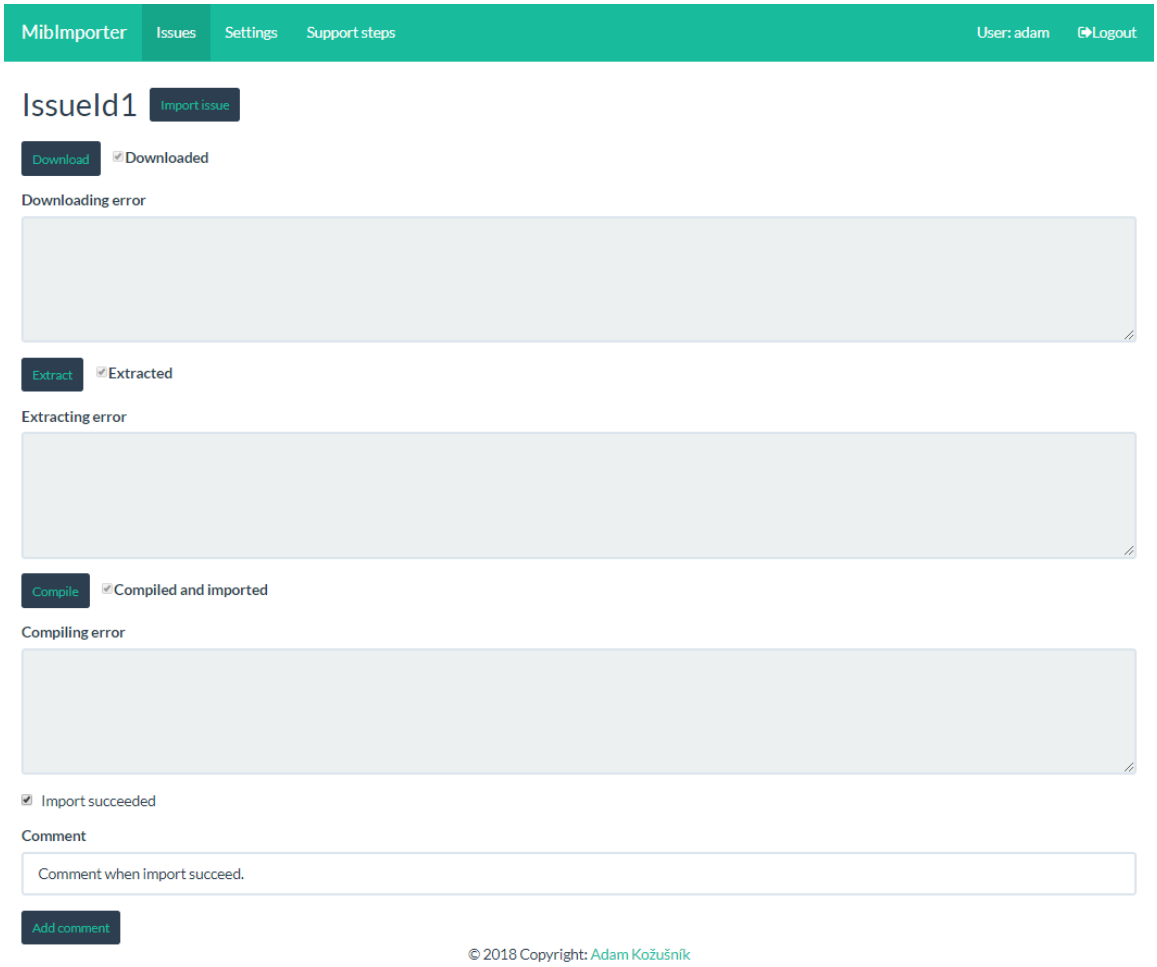
Obrázek 5.1: Webová stránka pro nastavení aplikace (snímek z vytvořené aplicace, zdroj vlastní)

Zpracování jednotlivých JIRA úkolů probíhá na záložce *Issues*, která komunikuje s kontrolérem *IssueController*. Ten umožňuje volání všech metod souvisejících se zpracováním JIRA úkolů. Právě v tomto kontroléru je držena jediná instance třídy *ProcessControl*, která je blíže popsána v kapitole 5.7.1. Vzhled záložky *Issue* můžeme vidět na obrázku 5.2. Po načtení této stránky jsou automaticky staženy všechny JIRA úkoly a zobrazeny v tabulce. Stažení úkolů se provede pouze při prvním načtení této stránky. V dalších načteních jsou už pouze načteny stažené JIRA úkoly. Nehrozí tak ztráta veškeré práce, pokud bychom pouze např. chtěli změnit nastavení aplikace. Pokud však chceme úkoly znovu stáhnout, je zde k dispozici tlačítko *Download issues*. Tlačítko *Import all issues* spustí proces zpracování všech JIRA úkolů. Pokud by došlo k nějaké neočekávané chybě, je tato chyba zobrazena do textového pole pod tímto tlačítkem. Po dokončení importu všech JIRA úkolů jsou aktualizovány hodnoty v tabulce. V tabulce jsou zobrazeny následující informace: identifikátor JIRA úkolu, identifikátor úkolu z oddělení pro podporu zákazníků, pravdivostní hodnoty informující o výsledku stažení archivů s MIB soubory, jejich extrakci a kompilaci. Pravdivostní hodnoty jsou zobrazovány pomocí komponenty *checkbox*. Na každém řádku tabulky se nachází také tlačítko *Import*, které uživatele přesměruje na stránku s detailem daného JIRA úkolu.



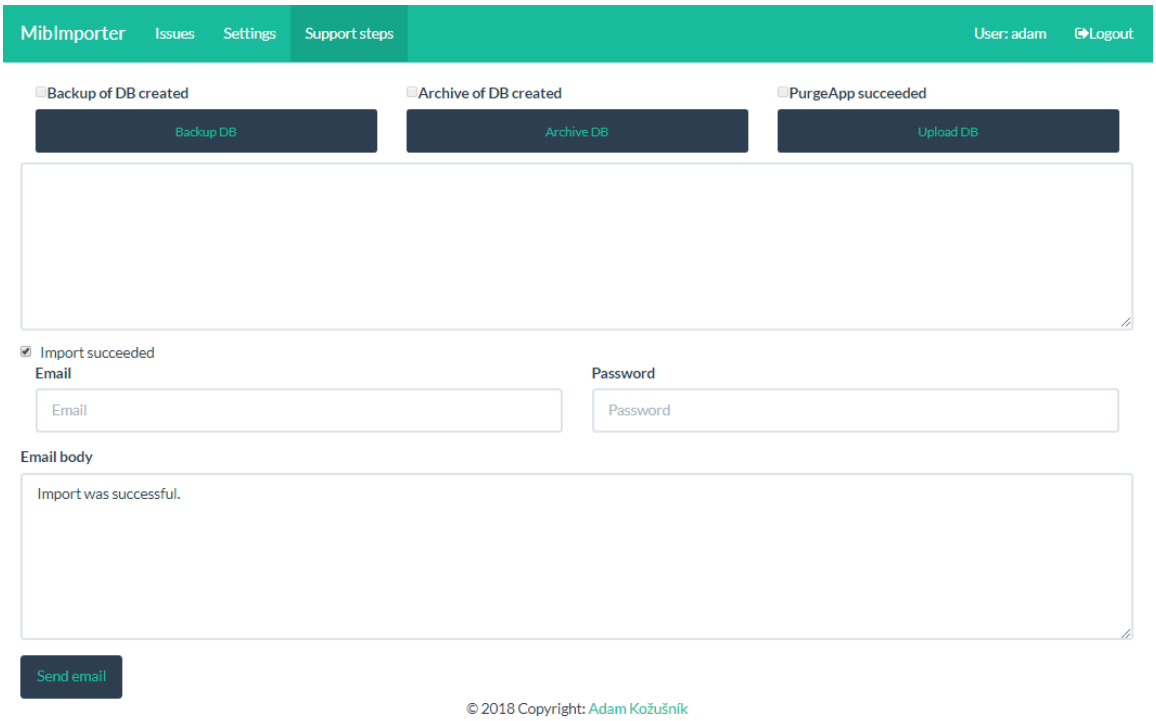
Obrázek 5.2: Webová stránka se všemi JIRA úkoly (snímek z vytvořené aplicace, zdroj vlastní)

Detail je možné vidět na obrázku 5.3. Na této stránce je možné spustit zpracování daného JIRA úkolu pomocí tlačítka *Import issue* v horní části. Dále jsou zde tlačítka pro jednotlivé kroky importu, tedy tlačítko *Download*, *Extract* a *Compile*. Po stisknutí těchto tlačítek se spustí adekvátní proces. Jakmile se proces dokončí, nastaví se podle navrácené hodnoty příslušný checkbox u daného tlačítka. V případě chyby je tato chyba zobrazena v příslušném textovém poli. Díky tomu uživatel ihned ví, co se stalo. Nakonec je zde možnost přidat komentář k úkolu v nástroji JIRA. Podle výsledku importu se nastaví checkbox *Import succeeded*, podle kterého se zobrazí šablona komentáře. Šablonu je možné zvolit také ručně zaškrtnutím/odškrtnutím uvedeného checkboxu. Šablonu lze také upravovat editací textu v textovém políčku *Comment*.



Obrázek 5.3: Detail vybraného JIRA úkolu (snímek z vytvořené aplicace, zdroj vlastní)

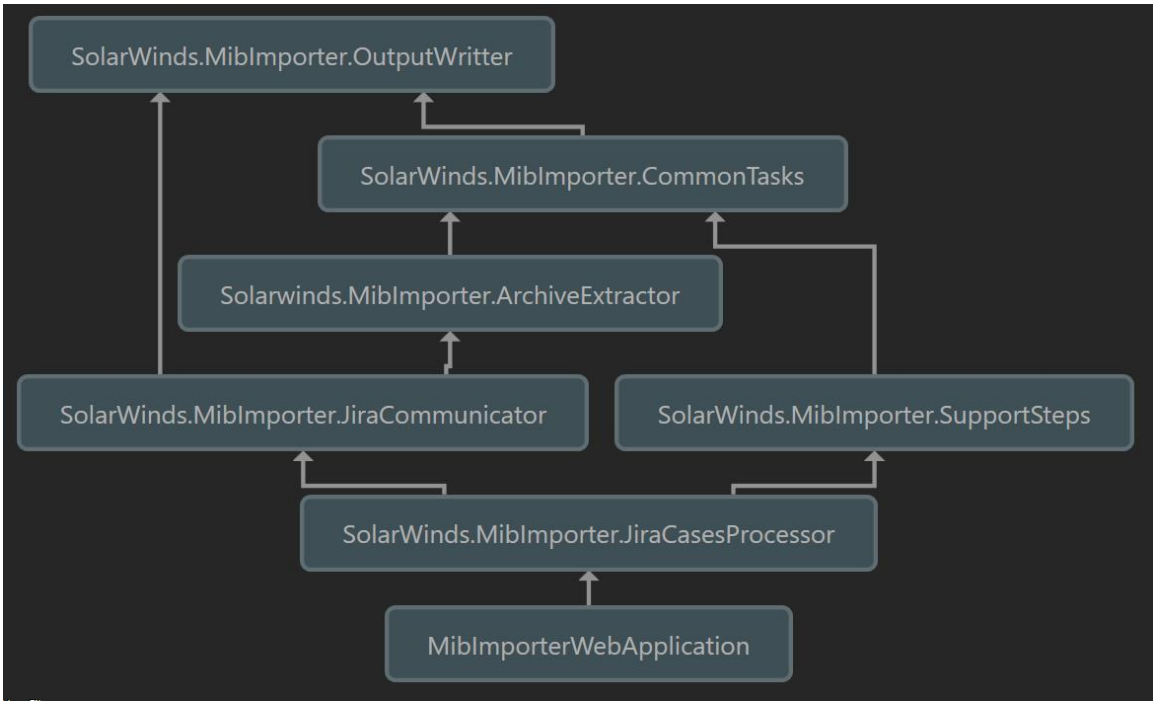
Jako poslední je zde záložka *Support steps*, kterou lze vidět na obrázku 5.4. V horní části stránky vidíme tlačítka *Backup DB*, *Archive DB* a *Upload DB* užívána pro zálohování databáze, vytvoření archivu databáze a nahrání databáze na server pro zákazníky. Nad každým tlačítkem je checkbox, který jako u zpracování JIRA úkolu informuje o úspěchu provedení dané akce. Pod tlačítky je dále komponenta *textarea*, kde se zobrazují případné chyby z uvedených akcí. Poté následuje formulář pro zaslání emailu. Pomocí checkboxu *Import succeeded* lze vybrat tělo emailu a to následně ještě upravit jako v případě komentáře u JIRA úkolu. K odeslání emailu je potřeba ještě zadat email odesílatele a heslo. Email se odešle stisknutím tlačítka *Send email*.



Obrázek 5.4: Stránka s funkcemi vykonávanými po zpracování všech JIRA úkolů (snímek z vytvořené aplicace, zdroj vlastní)

5.9 Diagram závislostí

Na obrázku níže 5.5 lze vidět diagram závislostí celého programu. Oproti diagramu uvedeném v kapitole návrhu 4.6 došlo k dosti velkým změnám. V původním návrhu se nepočítalo, že třída pro odesílání emailu bude v jednom modulu spolu se třídami pro archivaci databáze, zálohování databáze a její nahrání na server. Tyto třídy byly seskupeny do jednoho modulu *SupportSteps*. Dále v původním návrhu je *ArchiveExtractor* uveden pouze jako samostatná třída. Nakonec však extrahování archivů taktéž vydalo na samostatný modul. Rovněž nebylo počítáno s modulem *OutputWritter*, který slouží k zapisování do logovacího souboru a do konzole. Všechny moduly tedy mají referenci na tento modul a mohou tak zapisovat do stejného logovacího souboru. Nakonec je zde přidán hlavní modul *MibImporterWebApplication*, který komunikuje s modulem *JiraCasesProcessor*, kde se nachází třída *MibImporterResult* z původního návrhu.



Obrázek 5.5: Diagram závislostí jednotlivých modulů (vygenerováno z vývojového prostředí Visual Studio)

5.10 Testování

Testování probíhalo průběžně při vývoji jednotlivých modulů. Některé moduly bylo možné otestovat mimo firmu SolarWinds. Jednalo se např. o moduly pro extrahování archivů a kompilaci archivů. Firma SolarWinds mi poskytla reálná data, která jim zákazníci zasílali, a také MIB databázi. Bylo tak možné odzkoušet extrahování archivů různých typů a různých úrovní zanoření. Tento modul byl poskytnut firmě SolarWinds před dokončením celého programu. Zaměstnanci jej celou dobu využívali a bylo možné jej odladit. Funguje tedy přesně tak, jak bylo požadováno. Pro simulování chyb při kompilaci stačilo nějaké chyby v MIB souborech vytvořit (např. odstranit, či přidat uvozovku u popisu objektu apod). Pro modul *ArchiveExtractor* byly napsány také automatické testy za použití knihovny *NUnit*[9] a knihovny *Moq*[10]. Knihovna *Moq* slouží k nahrazení příslušné třídy třídou, která pouze vrací předem nastavené hodnoty. Například pokud u třídy *Extractor* zavoláme metodu pro extrahování všech archivů, můžeme předem definovat, jaký výsledek metoda vrátí. Přitom se samotná metoda vůbec neprovede. Pouze vrátí výsledek. Výsledky lze také upřesňovat podle parametrů předávaných do metod. Nakonec se výsledek metod porovnává s očekávaným výsledkem. Pokud jsou oba výsledky stejné, test projde. V opačném případě test selže. Díky testům bylo možné snadno odhalit chyby při změně kódu. Stačilo pouze testy spustit a během pár okamžiků výsledky ukázaly, zda vše stále funguje.

U dalších modulů však bylo testování komplikovanější, neboť jsem neměl k dispozici připojení do sítě firmy SolarWinds. K tomu by bylo zapotřebí připojení do jejich privátní sítě. Dále by však byly nutné přihlašovací údaje. Ty mi však nemohly být poskytnuty, neboť nejsem zaměstnancem této firmy. Testování tedy probíhalo v rámci konzultací s pracovníky firmy SolarWinds, a to na reálných JIRA úkolech. Drobné chyby se opravovaly přímo při konzultaci. Další úpravy probíhaly na základě informací v logovacích souborech. Díky nim bylo možné zjistit co, kde, jak a proč se pokazilo. Nejvíce problémů se odhalilo při spojení všech modulů dohromady a také při výměně nástroje *NetSuite* za *SalesForce*. Tehdy totiž došlo ke změně identifikátorů JIRA úkolů a nevytvářely se tak správně adresáře pro archivy s MIB soubory.

Pro testování webového rozhraní nebylo nutné mít přístup do sítě firmy SolarWinds. Většinu věcí bylo možné otestovat samostatně. K testování jsem používal také program *Postman*[11], který umožňuje zasílání webových požadavků ve zvoleném formátu na backend. Takto je možné otestovat, že komunikace funguje správně a backend vrací data v očekávaném formátu. Pro zobrazování byla vytvořena fiktivní data. Tato data obsahovala kombinace různých možností zobrazování tak, aby bylo ověřeno, že fungují všechny komponenty na webových stránkách korektně. Testováno bylo také zachytávání výjimek. Všechny metody v kontrolérech obsahují klauzuli *try/catch*, aby při náhlé chybě nedošlo k pádu programu. Chyba musela být zachycena a předána uživateli. Během testování byl projekt spouštěn výhradně z vývojového prostředí *Visual Studio* společnosti *Microsoft*. Takto byl vždy nastartován backend celé aplikace. Frontend byl spouštěn pomocí příkazu *ng serve*. Tento příkaz umožňuje upravovat kód webových stránek (soubory html a angularu) za běhu programu. Stačilo pouze uložit změny v souborech a došlo k automatickému překompilování kódu a projevení změn ve webovém prohlížeči. Tento způsob kompilace je také mnohem rychlejší, bylo tak možné vyvíjet webové rozhraní efektivněji.

[9] http://nunit.org/
[10] https://www.nuget.org/packages/Moq/4.8.2
[11] https://www.getpostman.com/

Níže jsou uvedeny testovací scénáře u vybraných akcí či modulů. Jedná se o stěžejní akce, které se provádějí při procesu importu MIB souborů. Tabulka 5.1 obsahuje testovací scénáře pro modul *JiraCommunicator*. Jedná se zejména testování přihlášení a následných prací s JIRA úkoly. Následující tabulka 5.2 popisuje testovací scénáře týkající se extrahování archivů. Poslední tabulka 5.3 zobrazuje scénáře týkající se kompilace MIB souborů a jejich vkládání do databáze.

Scénář	**Podmínky**	**Očekávaný výstup**	**Výsledek**
Přihlášení do nástroje JIRA	Počítač je připojen do privátní sítě firmy SolarWinds, přihlašovací údaje jsou správné	Dojde k přihlášení uživatele	OK
Stažení JIRA úkolů	Uživatel je přihlášen do nástroje JIRA	JIRA úkoly jsou staženy a zobrazeny uživateli, status úkolů se změní na: In progress	OK
Stažení archivů ze sdíleného úložiště	Uživatel je přihlášen do nástroje JIRA, archivy jsou v adresáři uvedeném v úkolu	Archivy jsou staženy do specifikovaného adresáře	OK
Adresář s archivy JIRA úkolu neexistuje	Adresář s archivy se ve sdíleném úložišti nevyskytuje	Navrácena chyba: Directory not found.	OK
Přidání komentáře k úkolu	Uživatel je přihlášen do nástroje JIRA, je vybrán konkrétní úkol	Komentář je přidán pod jménem přihlášeného uživatele	OK

Tabulka 5.1: Testovací scénáře pro modul *JiraCommunicator*

Scénář	Podmínky	Očekávaný výstup	Výsledek
Program 7zip nelze na počítači nalézt	Program 7zip není nainstalován	Navrácena chyba: Program 7zip is not installed.	OK
Extrahování archivu se známou příponou	Přípona archivu je zapsána mezi podporovanými příponami v konfiguračním souboru	Archiv je extrahován a přidán do seznamu extrahovaných archivů	OK
Extrahování archivu s neznámou příponou	Přípona archivu není mezi podporovanými ani ignorovanými příponami v konfiguračním souboru	Archiv není extrahován a je přidán do seznamu neznámých souborů	OK
Přípona archivu neodpovídá komprimačnímu algoritmu	Archiv korektně zkomprimován, jeho přípona je změněna	Archiv je extrahován se zprávou, že se jej nepodařilo extrahovat podle přípony, ale podle odpovídajícího komprimačního algoritmu	OK
Archiv je rozdělen do několika částí	Archiv je korektně extrahován a při komprimaci rozdělen na části	Archiv se nepodařilo extrahovat	OK

Tabulka 5.2: Testovací scénáře pro modul *ArchiveExtractor*

Scénář	Podmínky	Očekávaný výstup	Výsledek
MIB databázi nelze nalézt	Cesta k MIB databázi v konfiguračním souboru není správná	Navrácena chyba: Can not find MIB database.	OK
Nesprávné přihlašovací údaje k databázi	Přihlašovací údaje v konfiguračním souboru nejsou správné	Navrácena chyba: Can not connect to MIB database.	OK
MIB soubor je již přidán v databázi	Kompilace je prováděna nad již zkompilovaným souborem	MIB soubor je zkompilován a znovu přidán do databáze	OK
MIB soubor není v databázi	MIB soubor neobsahuje chyby	Soubor je přidán do databáze	OK
MIB soubor obsahuje chyby	Program Notepad++ je nainstalován	Soubor je otevřen v programu Notepad++ na řádku s chybou	OK

Tabulka 5.3: Testovací scénáře pro kompilaci MIB souborů

5.11 Spuštění

Jak bylo uvedeno v kapitole 4, program se měl spouštět také jako služba v naplánovanou dobu. Od tohoto řešení však bylo ustoupeno. Tím hlavním důvodem je nutnost komunikovat s nástrojem JIRA. To však vyžaduje, aby byl některý uživatel přihlášen. Pokud se totiž uživatel nepřihlásí, nelze stáhnout JIRA úkoly, a tudíž ani stáhnout archivy s MIB soubory ze sdíleného úložiště. Aby bylo možné program spustit jako službu, musely by se uchovávat přihlašovací údaje některého ze zaměstnanců, což rozhodně není žádoucí.

Aplikace byla přeložena pomocí příkazu *ng serve –publish*. Tím došlo k vytvoření zdrojových knihoven pro běh v *Internet Information Services* (dále již IIS). Poté v IIS byl vytvořen webový server, jemuž se jako zdrojový adresář předal adresář, kde byly zdrojové knihovny přeloženy. Dále byl nastaven příslušný název webové stránky a službu bylo možné spustit zadáním názvu stránky do internetového vyhledávače. Aplikace tak běží na lokálním počítači, který je určen pouze ke zpracování MIB souborů. Zaměstnanci se tak musí nejprve vzdáleně připojit na tento počítač a až poté mohou spustit aplikaci pro importování MIB souborů.

Kapitola 6

Závěr

V rámci této práce se povedlo detailně analyzovat problematiku importu MIB souborů ve firmě SolarWinds. Pro pochopení dané problematiky bylo potřeba nastudovat prostředí, ve kterém se automatizace provádí, a také používané protokoly. Po seznámení se s prostředím došlo na zkoumání celého procesu importu MIB souborů. Bylo potřeba se důkladně seznámit s prostředky, které firma SolarWinds již měla. Jednalo se o nástroje, které umožňovaly provést některé kroky celého procesu. Některé z těchto nástrojů fungovaly správně, jiné bohužel pracovaly špatně a bylo potřeba je upravit. Nějaké nástroje však bylo nutné nahradit novými, lépe fungujícími, neboť jejich úprava by byla náročná.

Na základě výše popsaného a konzultací se zaměstnanci firmy SolarWinds došlo k přesné specifikaci požadavků na automatizaci procesu importu MIB souborů. Všechny požadavky byly vzaty v úvahu při návrhu celé architektury systému, který automatizaci provádí. Došlo k vybrání vhodných technologií, nástrojů a postupů, které se při vývoji uplatnily. Návrh byl vypracován nejen pro logiku celého procesu, ale také pro grafické uživatelské rozhraní v podobě webových stránek. Rovněž byly navrženy změny pro jednotlivé existující nástroje.

Na základě analýzy existujících nástrojů byly provedeny jejich úpravy. Grafika webových stránek doznala několika změn, zejména designu. Implementace probíhala za neustálého konzultování se zaměstnanci firmy SolarWinds. Bylo tak možné rychle zakomponovat měnící se požadavky. Navržený systém se povedlo úspěšně naimplementovat a splnit všechny zadané požadavky. Tento nástroj pomohl výrazně urychlit importování MIB souborů. Manuální zásahy uživatele byly zachovány tak, jak se předpokládalo. Při importu je totiž nutné opravovat poškozené či chybné MIB soubory. Celý proces však lze spustit zcela samostatně a dojde tak ke zpracování všech úkolů, které neobsahují poškozené MIB soubory. Uživatel se tak bude věnovat pouze těmto úkolům, což také velmi šetří čas potřebný pro celý import. Jednotlivé úkoly je rovněž možné zpracovávat samostatně, dokonce jejich jednotlivé kroky lze provádět odděleně. Akce jako zálohování databáze, či její nahrání na server pro zákazníky a další, se rovněž podařilo úspěšně implementovat. Vše se přehledně zobrazuje uživateli, a ten je tak informován o stavu jednotlivých úkolů a celého procesu.

Ideálním vylepšením by byla úprava kompilátoru, kdy by kompilátor byl sám schopen opravit alespoň část chyb. Chyby v MIB souborech se velmi často opakují. Jedná se často o špatně dané uvozovky, či nevhodně danou mezeru. Tato úprava by však byla velmi náročná. Kompilátor by také mohl být upraven tak, aby se pokoušel importovat pouze MIB soubory, které do databáze ještě importovány nebyly. Bylo by nutné si pamatovat soubory, které již importovány byly a kompilátoru předat pouze ty, které ještě importovány nebyly.

Literatura

[1] Jira Software. 2018, [Online; navštíveno 8.1.2018].
URL https://www.atlassian.com/software/jira

[2] CASE, J.; MCCLOGHRIE, K.; ROSE, M.; aj.: Structure of Management Information Version 2 (SMIv2). 1999, [Online; navštíveno 9.1.2018].
URL https://tools.ietf.org/html/rfc2578#page-19

[3] DENHARTOG, M.: Demystifying the SNMP MIB. únor 2008, [Online; navštíveno 9.1.2018].
URL https://scadahacker.com/library/Documents/ICS_Protocols/Demystifying%20the%20SNMP%20MIB.pdf

[4] HELD, G.: *LAN Management with SNMP and RMON*. Wiley Computer Publishing, 1996.

[5] Mauro, D. R.; SCHMIDT, K. J.: *Essential SNMP*. O'Reilly & Associates, 2001, ISBN 0-596-00020-0.

[6] MuleSoft: What is a REST API? [Online; navštíveno 11.1.2018].
URL https://www.mulesoft.com/resources/api/what-is-rest-api-design

Příloha A

Obsah CD

Přiložené CD obsahuje:

- /src - adresář se zdrojovými kódy
- /text - tuto technickou zprávu ve formátu PDF a zdrojové kódy této technické zprávy
- README.txt - textový soubor s návodem pro spuštění